

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального
назначения»
Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

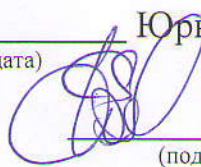
Web-сервис электронной коммерции на базе бессерверной
архитектуры

Студент


(подпись, дата)

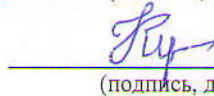
Юрьев Владимир Александрович
(ФИО полностью)

Руководитель


(подпись, дата)

Федюнин Р.Н.
(фамилия, инициалы)

Нормоконтролёр


(подпись, дата)

Кучин А.В.
(фамилия, инициалы)

Рецензент

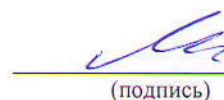
(должность, место работы)


(подпись, дата)

Васильков А.В.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 10.06.2020 г. № 13)

Заведующий кафедрой


(подпись)

Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
18.06.2020 № 1)

Секретарь ГЭК


(подпись)

Гурин Е.И.
(фамилия, инициалы)

Пенза 2020

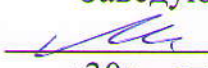
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»
Заведующий кафедрой
 Митрохин М.А.

«30» апреля 2020г.

**ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ**

1. Студент Юрьев Владимир Александрович гр. 15BO2 факультета вычислительной техники.

2. Тема работы: «Web-сервис электронной коммерции на базе бессерверной архитектуры».

Тема утверждена приказом ПГУ № 0335а/о от 30.04.2020 г.

3. Руководитель работы к.т.н., доцент каф. ВТ ПГУ Федюнин Р.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.).

Назначение разработки – Система предназначена для обработки и передачи сообщений с помощью различных протоколов (HTTP/HTTPS, AWS Lambda, Email). В системе необходимо предусмотреть работу с WEB интерфейсом, облачными системами Amazon. Система должна работать с нагрузкой не менее 200 одновременно работающих клиентов.

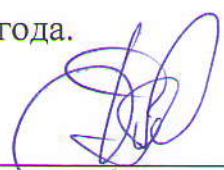
5. Перечень подлежащих разработке вопросов.

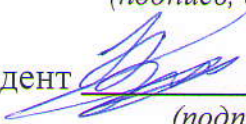
Требуется произвести анализ предметной области, выбрать функциональный состав системы, осуществить выбор стека технологий для реализации системы, разработать алгоритм работы системы, разработать структурную схему системы, разработать диаграммы: прецедентов, активностей, последовательностей, объектов, классов, с последующим написанием программы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	30.04.20	
2.	Выбор компонентов для реализации принципиальной схемы	10	07.05.20	
3.	Составление принципиальной схемы устройства	20	08.05.20	
4.	Написание программы-прошивки для устройства	10	14.05.20	
5.	Отладка системы	10	16.05.20	
6.	Моделирование и поиск уязвимостей, исправление ошибок в работе системы	10	22.05.20	
7.	Оформление выпускной работы	30	28.05.20	

Дата выдачи задания 30 «апреля» 2020 года.

Руководитель дипломного проекта  Федюнин Р.Н.
(подпись, дата)

Задание к исполнению принял студент  Юрьев В.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Дипломная работа содержит 138 листов, 48 рисунков, 5 источников, 3 приложения.

ОБЛАЧНАЯ СИСТЕМА, КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ, СЕРВИСНЫЕ ФУНКЦИИ, БЕССЕРВЕРНАЯ АРХИТЕКТУРА.

Цель работы – разработать масштабируемую, отказоустойчивую систему продажи товаров на базе облачных технологий.

Объектом разработки является программное приложение, обеспечивающее клиентам доступ к каталогу товаров и услуг, администраторам доступ к настройкам системы, а программистам к инструментам добавления нового функционала на базе облачного сервиса.

В результате проведенной работы должен быть разработан проект, удовлетворяющий поставленной задаче, а именно, создан программный комплекс – «тонкий» клиент, бессерверная бизнес логика и хранилище, и разработаны соответствующие алгоритмы, реализующие возможность хранить, обрабатывать и доставлять до потребителя информацию в виде сообщений.

Данное приложение рекомендуется использовать для массового продвижения товаров и услуг. Система может быть внедрена в сегменте электронной коммерции компании сегмента масс-маркет.

					ПГУ 1.090501.17.001 ПЗ					
Изм.	Лист	№ докум.	Подпись	Дата	WEB сервис электронной коммерции на базе бессерверной архитектуры Пояснительная записка			Лит.	Лист	Листов
Разраб.		Юрьев В.А.								
Провер.		Федюнин Р.Н.							2	138
Н. Контр		Кучин А.В.						ФВТ, каф. ВТ гр. 15В02		
.										
Утверд.										

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	7
1.1 Сбор материала по теме.....	7
1.2 Исследование облачных сервисов бессерверных вычислений	12
1.3 стек технологий и алгоритмы реализации системы	16
1.3.1 Выбор инструментов разработки	16
1.3.2 Выбор инструментов разработки	17
1.4 Реализация механизмов работы с файлами	25
1.5 Реализация механизмов работы с файлами	26
2 РАЗРАБОТКА ПРИЛОЖЕНИЯ	27
2.1 Разработка сервисов обработки GET и POST запросов приложения..	28
2.2 Разработка обработчиков для СУБД.....	36
2.3 Разработка СУБД	48
2.4 Интерфейсы приложения	52
3 ТЕХНОЛОГИЯ РАЗРАБОТКИ ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ	64
4 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ ИНВЕСТИЦИЙ ВО ВНЕДРЕНИЕ WEB СЕРВИСА ЭЛЕКТРОННОЙ КОММЕРЦИИ НА БАЗЕ БЕССЕРВЕРНОЙ АРХИТЕКТУРЫ.....	73
4.1 Обоснование выбора объекта	73
4.2 Расчет затрат на модернизацию WEB сервиса и покупку лицензий...	75
4.2.1 Расчет затрат на оплату труда.....	75
4.2.2 Расчет отчислений на социальные нужды.....	76
4.2.3 Расчет накладных расходов	77
4.2.4 Расчет амортизационных отчислений.....	77
4.2.5 Расчет затрат на обслуживание оборудования	78
4.2.6 Расчет затрат на электроэнергию	78
4.2.7 Расчет стоимости модулей сервиса «Amazon».....	79

4.2.8 Расчет цены программы	80
4.3 Расчет эксплуатационных расходов.....	80
4.4 Расчет денежного потока.....	82
5 ПСИХОФИЗИОЛОГИЧЕСКИЕ ФАКТОРЫ, ВОЗДЕЙСТВУЮЩИЕ НА ОПЕРАТОРА	86
5.1 Физические факторы.....	86
5.2 Тяжесть и напряженность труда.....	88
5.3 Психологические факторы	88
5.5 Влияние режима работы на оператора ЭВМ.....	96
5.6 Выводы и рекомендации	98
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	99
ПРИЛОЖЕНИЕ А	100
ПРИЛОЖЕНИЕ В	118
ПРИЛОЖЕНИЕ С	124

ВВЕДЕНИЕ

Сегодня компании стали активно обращать внимание на «облака», думать об оптимизации и уплотнении вычислений на своих и облачных ресурсах, рассматривать пути сокращения расходов на ресурсы для вычислений.

С развитием облачных сервисов вычисления становятся все более похожи на услугу, а потребителей (разработчиков) перестает интересовать физическая серверная инфраструктура, которая выполняет вычисления.

Говоря о бессерверных вычислениях (serverless computing), нужно понимать, что на самом деле без физических и виртуальных серверов не обойтись ни при каких условиях, значит, речь идет не об отказе от аппаратной части, а о новом подходе к предоставлению сервисов в виде программных функций.

Если задача команды разработчиков - сосредоточиться на бизнес-логике и выпускать продукты на рынок со скоростью света, то реализация проекта в рамках облачных сервисов отличный инструмент.

Так и появляются на рынке бессерверные вычисления. Serverless в данном случае означает отсутствие задач/проблем по управлению инфраструктурой.

Идея такого подхода в том, что логика приложения разбивается на независимые функции. Они имеют событийную структуру. Каждая из функций выполняет одну микрозадачу. Все, что требуется от разработчика, — загрузить функции в предоставленную консоль и соотнести их с источниками событий.

Впервые такие сервисы FaaS были реализованы в 2014 году в проекте Open-Source Microservice Hosting Platform. Вскоре появились сервисы

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

Amazon AWS Lambda, в 2016 году — Google Cloud Functions, Microsoft Azure Functions, IBM/Apache's OpenWhisk и в 2017-м — Oracle Cloud Fn.

Форма предоставления ПО как сервисов — Software as a Service (SaaS) — за годы своего существования серьезно изменилась. На сегодняшний момент она имеет три лица: контейнеры (CaaS), приложения (PaaS), функции (FaaS).

Задача производственной практики выбрать оптимальную платформу для реализации приложения электронной коммерции на базе бессерверной технологии, разработать архитектуру приложения, описать реализуемые приложением функции, нарисовать шаблоны приложения и составить uml диаграммы работы приложения.

Основными критериями выбора инструментов реализации будут – легкость регистрации и доступа, наличие у сервиса инструментов реализации бизнес логики (триггеры реакции) и хранения информации, а также максимально возможная поддержка современных языков для web разработки.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Сбор материала по теме

В данном разделе будут рассмотрены подходы для реализации системы электронной коммерции с точки зрения выбора архитектуры и стека технологий реализации приложения (рисунок 1.1).

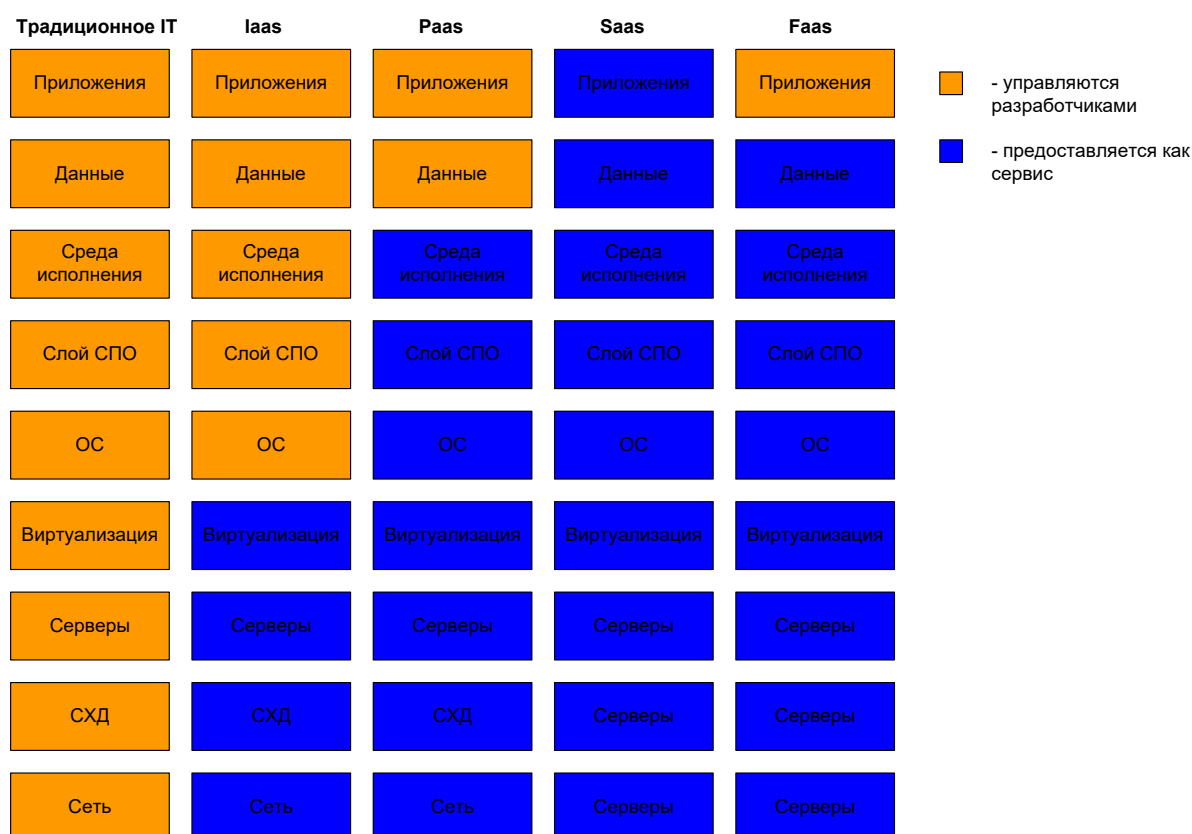


Рисунок 1.1 – Эволюция реализации и развертывания клиент серверных приложений

Примитивным подходом для реализации приложения с «тонким» клиентом в области электронной коммерции является выбор трехзвенной архитектуры с фактической реализацией приложения в рамках паттерна Модель – Вид – Контроллер на базе своих собственных аппаратных серверов или арендованной серверной инфраструктуры.

Стеком технологий в данном случае будет: LAMP/WAMP: Linux/Windows-операционная система; Apache - веб-сервер; MariaDB/MySQL-СУБД; PHP-язык программирования, используемый для создания веб-приложений (помимо PHP могут подразумеваться другие языки, такие как Perl и Python).

Сама система будет выглядеть в виде трехзвенной архитектуры, показанной на рисунке 1.2. Как видно из схемы на рисунке 1.2. приложение реализовано в виде трех слоев, каждый из которых связан с соседним слоем приложения.

Слой представления реализован в виде роутинга - маршрутизации: входящий URL разбирается специальным образом и по его результату выполняется определенный код уровня бизнес логики. С роутингом напрямую связано понятие ЧПУ (человеко-понятные url), которое позволяет исключить в адресах сложные параметры.

В данном приложении (рисунок 1.2) видна реализация отдельных маршрутов для товара, заказов и пользователей. У приложения также предусмотрены маршруты для чат-бота и обработчика платежей.

Все маршруты будут вызывать функции-обработчики на бизнес-уровне, и обработанные данные бизнес уровня будут отправлены на уровень данных - база данных (СУБД) и хранение файлов и изображений.

Такой подход идеально подходит для любого небольшого приложения. Это будет нормально работать до тех пор, пока количество заказов на товар или услугу не вырастут до определенного уровня. Тогда возникнет необходимость масштабировать свою инфраструктуру. Для масштабирования монолитного приложения, необходимо отсоединить слой данных (рисунок 1.3).

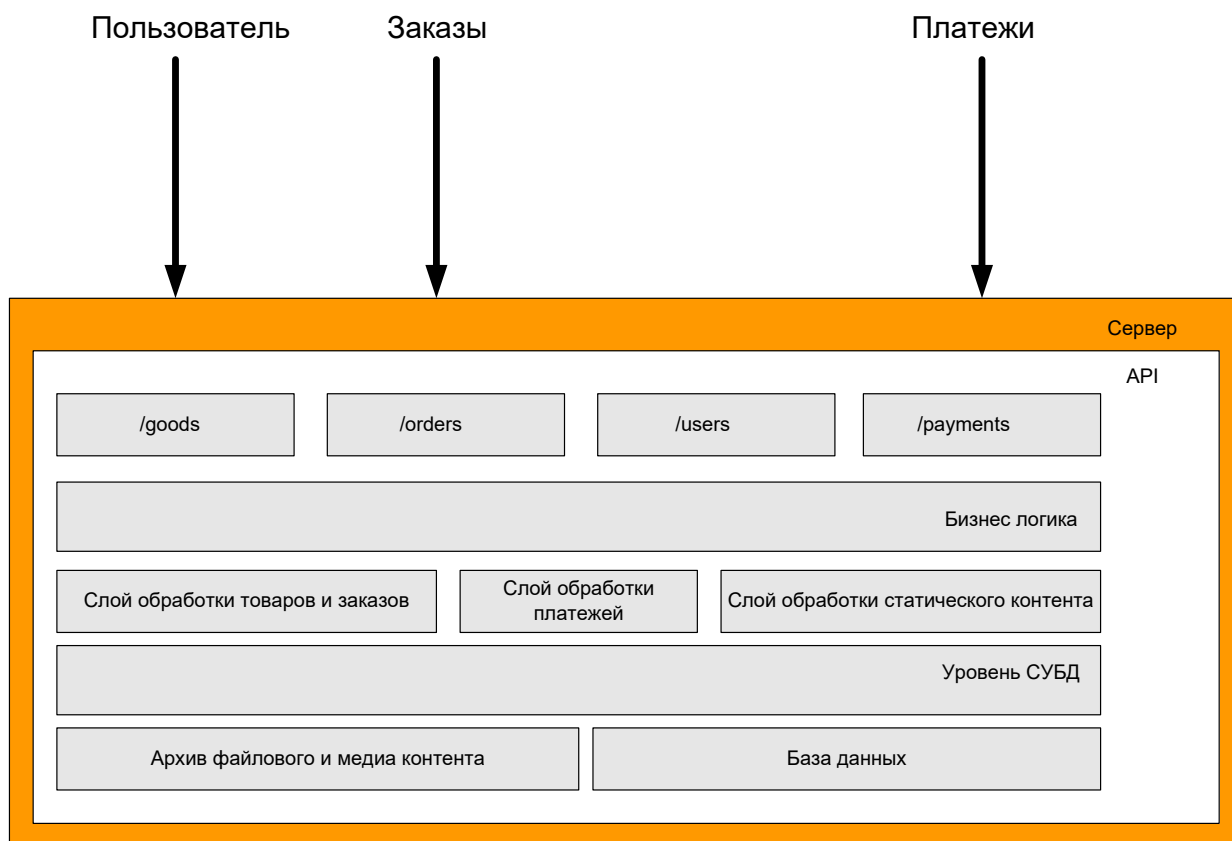


Рисунок 1.2 – Классическая трехзвенная архитектура приложения

После этого приложение будет выглядеть так, как показано на рисунке 1.3. В образованной архитектуре сохраняется еще один монолитный блок: вид/отображение, API-маршруты (роутинг) и бизнес-логика. Но СУБД уже может быть распределенным объектом, который имеет запас прочности при интенсивных нагрузках на приложение. Но в данном подходе все равно сохраняется необходимость иметь штат инженеров и производить постоянное обслуживание и апгрейд аппаратной части.

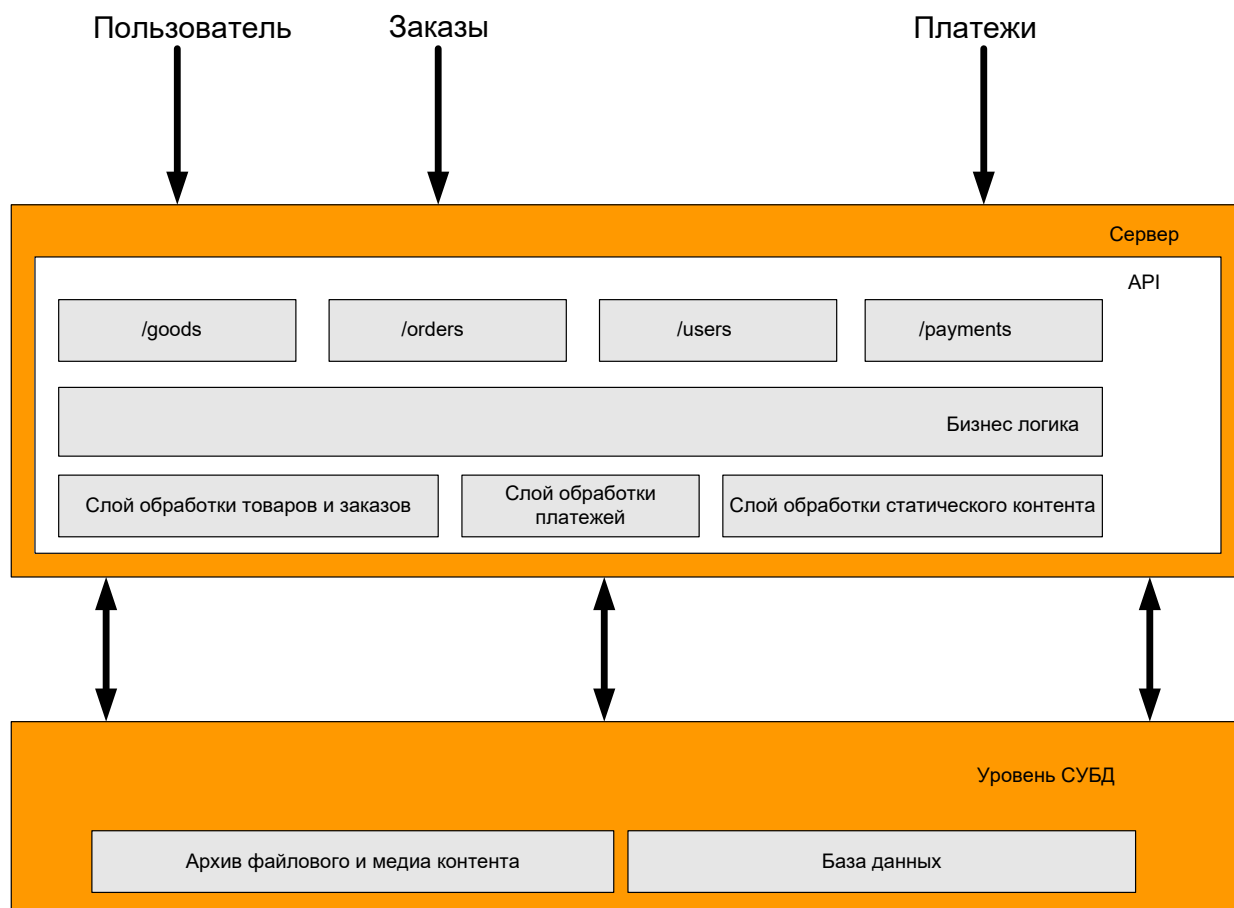


Рисунок 1.3 – Минимальная архитектура распределенного трехзвенного приложения

Во избежание постоянных затрат на обслуживание инфраструктуры и для увеличения «прочности» приложения к нагрузкам со стороны конечных пользователей имеет смысл рассмотреть возможность реализации приложения на базе облачных (бессерверных) технологий (рисунок 1.4).

В основе бессерверного приложения лежат два фундаментальных момента: событийная обработка запросов и функций и распределенность системы.

Фактически вместо одного (рисунок 1.2) или нескольких серверов (рисунок 1.3) разработчики получают множество серверов и контейнеров с изолированным друг от друга контентом или логикой (рисунок 1.4).

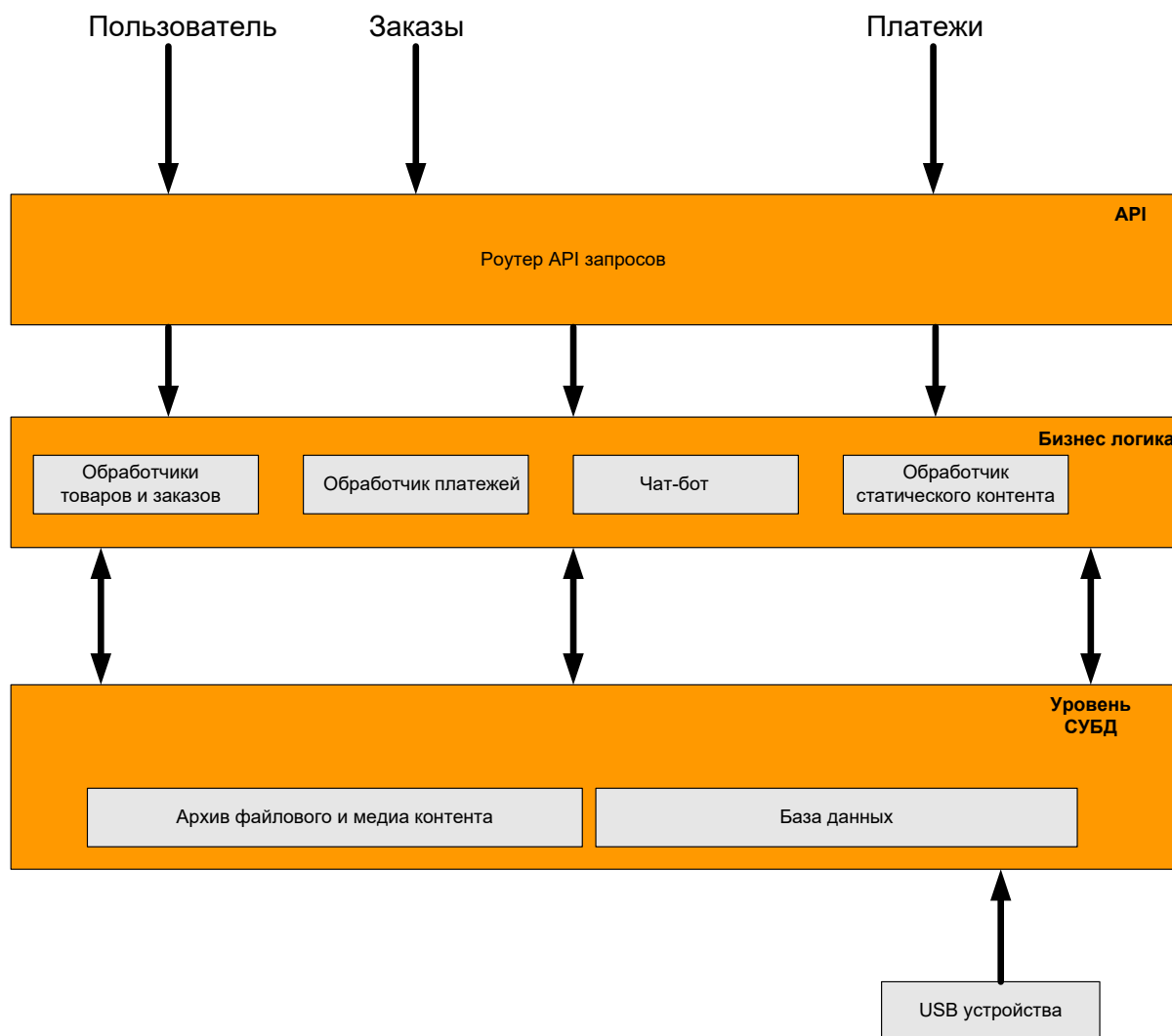


Рисунок 1.4 – Архитектура бессерверного приложения.

В бессерверном приложении запросы обрабатываются API маршрутизатором, который анализирует их и отправляет на уровень бизнес логики. Бизнес логика представлена набором функций, каждая из которых реализована в виде отдельной услуги. Уровень данных также представлен услугой, данные в которую могут поступать как через канал приложения в виде запросов на чтение и запись, так и в виде доступа к файловой системе (как показано на рисунке 1.4 сохранение фотографий возможно прямым доступом к хранилищу данных).

Таким образом, приходим к выводу, что для реализации современного приложения электронной коммерции, целесообразно выбрать

инструментарий облачных вычислений, из-за отсутствия штата системных администраторов и собственных или арендованных физических серверов.

В следующем разделе будут рассмотрены популярные облачные сервисы бессерверных вычислений.

Исследуем достоинства и недостатки данного подхода и выберем инструмент для реализации функционального ядра проекта, с точки зрения программиста.

1.2 Исследование облачных сервисов бессерверных вычислений

Существует много критериев, по которым можно сравнить основные сервисы предоставления облачных вычислений и бессерверных услуг.

Выделим основные из них: план оплаты услуг, средняя цена за месяц использования, Service Level Agreement (SLA) – соглашения по уровню сервиса (вне зависимости от предыдущей производительности) в процентах, количество датацентров и геопривязка, масштабирование (вертикальное/горизонтальное), техническая поддержка, мониторинг, API и языки программирования, стоимость трафика (таблица 1.1).

Изм.		Таблица 1.1 – Сравнение облачных сервисов Amazon, Azure, Google			
Лист					
№ докум.					
Подпись					
Дата					
ПГУ 1.090501.17.001 ПЗ					
13	Лист				

	Amazon	Azure	Google
1	2	3	4
API маршрутизатор, Обработчики	AWS Lambda	Azure Functions	Google Cloud Functions
Цена выполнения функции API маршрутизатором	первый миллион запросов за месяц бесплатно сверх – 20 центов за каждый миллион запросов	первый миллион запросов за месяц бесплатно сверх – 20 центов за каждый миллион запросов	первые два миллиона запросов за месяц бесплатно сверх – 40 центов за каждый миллион запросов
Набор триггеров для маршрутизации запросов	Amazon S3 - файловый хостинг. Amazon DynamoDB - система управления базами данных класса NoSQL Amazon Kinesis Stream - сервис для потоковой передачи данных в режиме реального времени Amazon Simple Notification Service - сервис отправки уведомлений между сетевыми компонентами Amazon Web Services и внешними компонентами	Azure Storage Azure Event Hubs Queues and Topics Azure Schedule HTTP (REST or WebHook)	Google Cloud Pub/Sub triggers HTTP

Изм.		Продолжение таблицы 1.1			
Лист					
№ докум.					
Подпись					
Дата					
ПГУ 1.090501.17.001 ПЗ					
Лист	14				
		1	2	3	4
			Amazon Simple Email Service - один из облачных веб-сервисов от компании Amazon Amazon Cognito AWS CloudFormation Amazon CloudWatch Logs Amazon CloudWatch Events AWS CodeCommit Scheduled Events AWS Config Amazon Alexa Amazon Lex Amazon API Gateway		
Набор средств разработки (SDK)		Android Browser iOS Java .NET Node.js	.NET Java Node.js PHP Python Ruby	Java Python Node.js Ruby GO .NET	

Продолжение таблицы 1.1

1	2	3	4
	Ruby Python PHP Go C++ AWS Mobile AWS IoT	Mobile Media Android iOS JavaScript Swift Windows	PHP

Сравнение характеристик трех лидеров рынка облачных и бессерверных вычислений показывает, что Amazon предоставляет качественный сервис, который в отличие от конкурентов обладает на сегодняшний момент большим функционалом, необходимым для реализации дипломного проекта.

Основными достоинствами решения Amazon являются:

1. простота создания и развёртывания продукта.
2. горизонтальное масштабирования и высокая доступность проекта из коробки.
3. оплата только за использованное время
4. интеграция со всеми известными сервисами СУБД, SNS и т.д., которые позволяют обработать технические и бизнес кейсы.

1.3 Стек технологий и алгоритмы реализации системы

1.3.1 Выбор инструментов разработки

Инструментарий был выбран исходя из возможностей выбранной платформы Amazon (таблица 1.2).

Таблица 1.2 – Инструменты разработки

№	Наименование инструмента	Причина выбора
1	2	3
1	Amazon	устойчивый сервис с поддержкой современных технологий
2	Node.js	программная платформа, превращающая JavaScript из узкоспециализированного языка в язык общего назначения.

Продолжение таблица 1.2

№	Наименование инструмента	Причина выбора
1	2	3
3	Claudia API Builder	библиотека расширений, которая помогает интегрировать Node.js с AWS API Gateway.
4	DynamoDB	система управления базами данных класса NoSQL в формате «ключ — значение», предлагаемая Amazon.com

Таким образом выбрали 4 инструмента: Amazon, Node.js, Claudia API Builder и DynamoDB.

1.3.2 Выбор инструментов разработки

Для разработки функций собственного API необходимо определиться с функционалом приложения (таблица 1.3).

Таблица 1.3 – Список задач, решаемых приложением

№	задачи приложения	маршрут роутинга
1	просмотр всего ассортимента товаров	GET/goods
2	получение заказов на товар	GET/orders
3	создание заказа на товар	POST/orders
4	обновление заказа на товар	PUT/orders
5	отмена заказа на товар	DELETE/orders

Исходя из данных таблицы 1.3 и организации проекта Node.js структура проекта выглядит следующим образом (рисунок 1.5).

Как показано на рисунке 1.5 все HTTP запросы от пользователей обрабатываются API Gateway платформы Amazon, которая берет на себя

функцию прокси сервера, сортирующего все запросы. Далее HTTP запросы прошедшие первичную проверку попадают в контейнер Claudia API Builder, который разбирает URL HTTP запроса и формирует запрос к конкретному обработчику, для которого данный HTTP запрос предназначен.

В данном конкретном случае имеется пять обработчиков, которые полностью покрывают возможные варианты обработки заказа товаров.

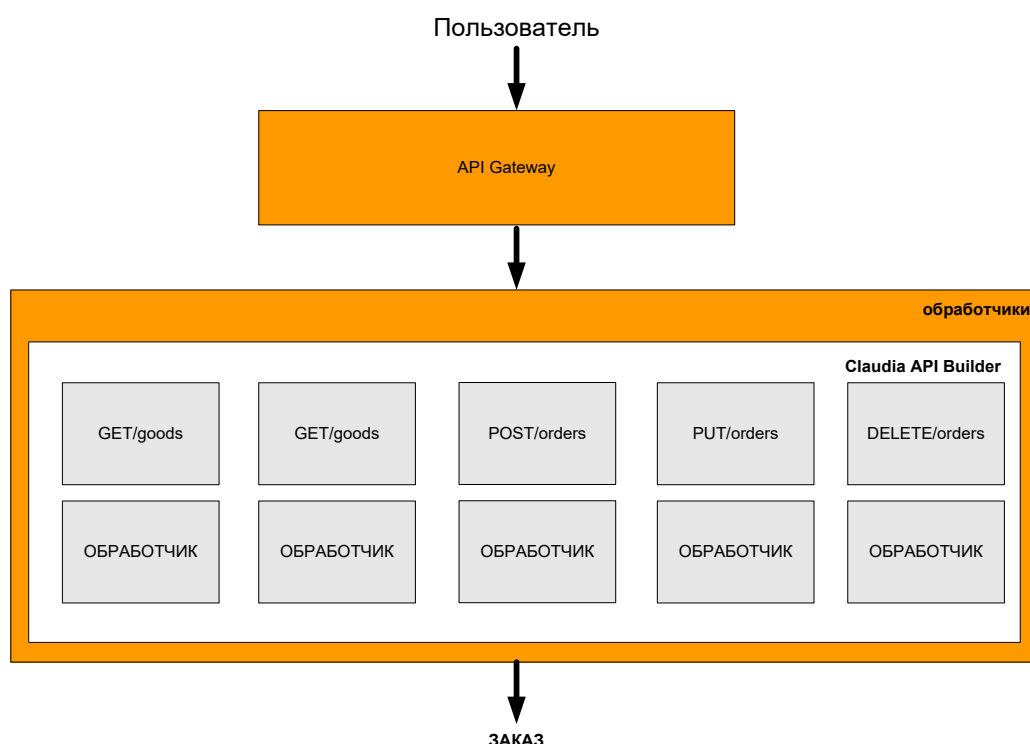


Рисунок 1.5 – Структура приложения

Сам обработчик в свою очередь представляет собой Amazon Lambda функции (рисунок 1.6). AWS Lambda - это сервис бессерверных вычислений, который запускает программный код в ответ на определенные события и отвечает за автоматическое выделение необходимых вычислительных ресурсов. В случае данного приложения триггером является HTTP запрос.

Каждая функция содержит пользовательский код и некоторые данные конфигурации, включая имя функции и требования к ресурсам. Функции Lambda не сохраняют состояние и никак не зависят от базовой

инфраструктуры, поэтому Lambda может быстро загрузить столько копий функции, сколько нужно для масштабирования в соответствии с количеством входящих событий. В общем случае обработчик включает код пользователя, API Amazon для обработки служебных запросов и API работы с СУБД Amazon DynamoDB.

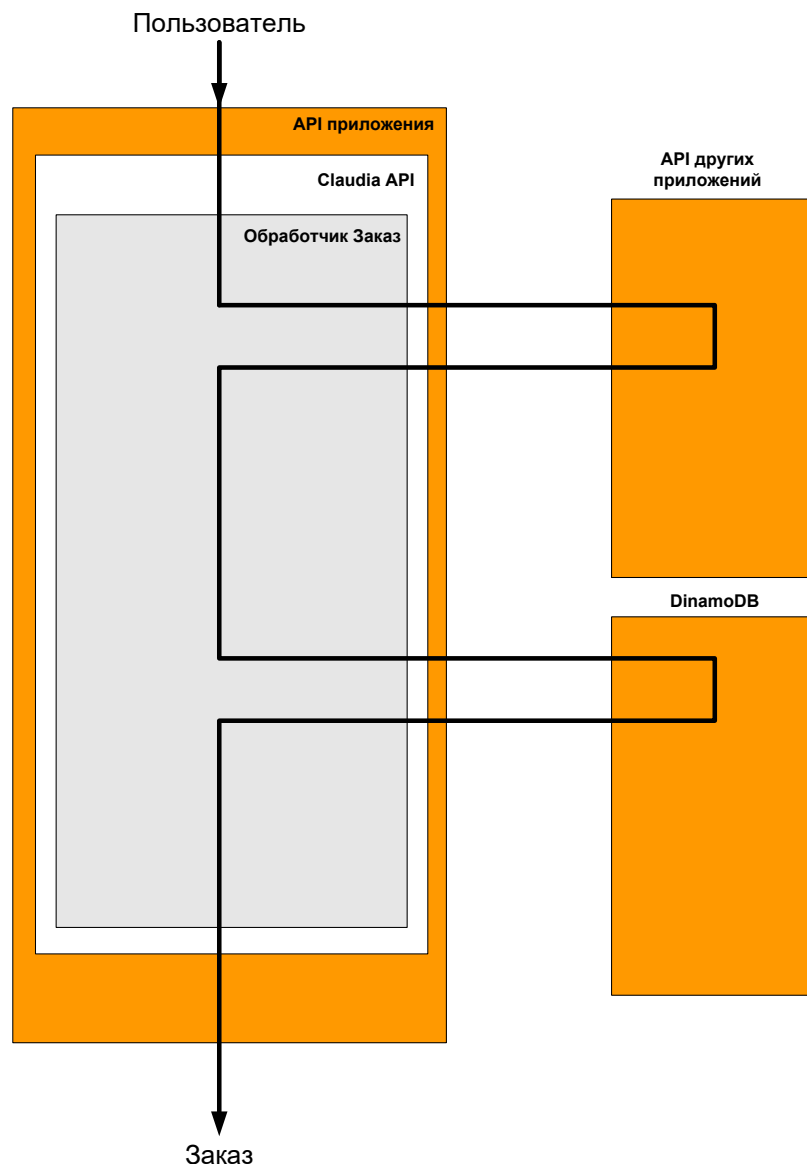


Рисунок 1.6 – Общая структура обработчиков

На рисунках 1.7 и 1.8 представлены конкретные примеры Amazon Lambda функций выполняющие задачи обработки заказа и его удаления из системы заказов.

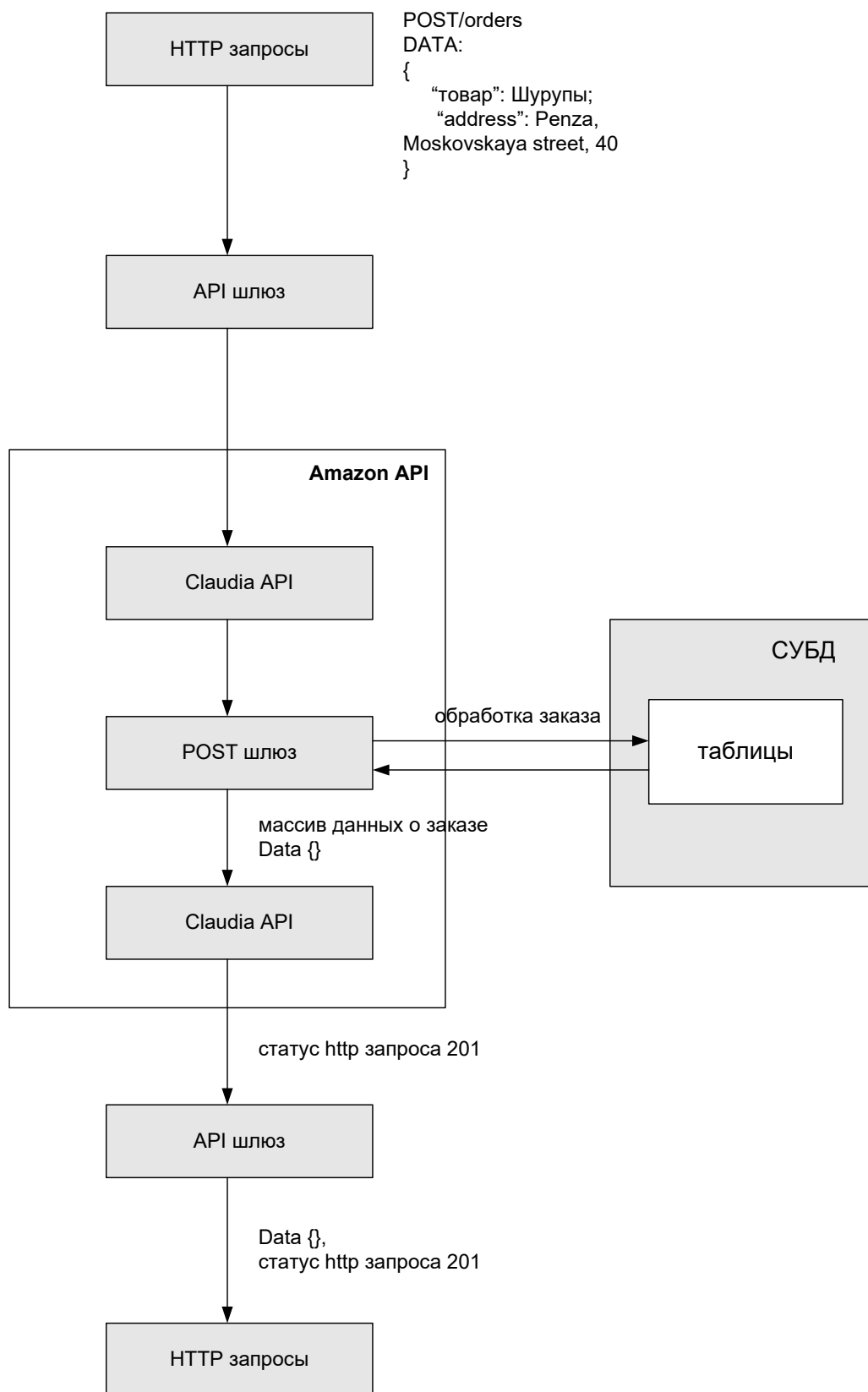


Рисунок 1.7 – Структура обработчика «обработка заказа»

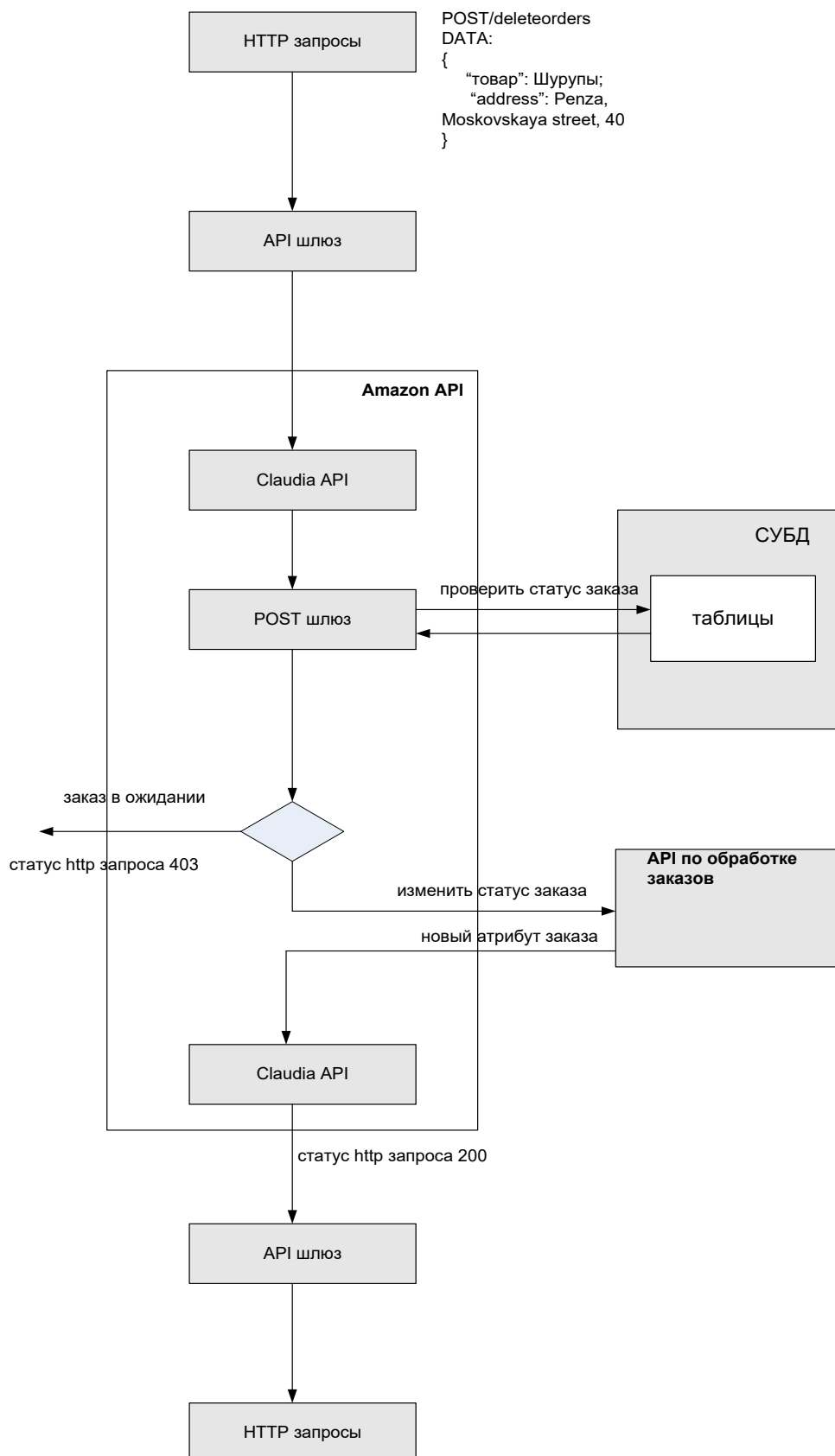


Рисунок 1.8 – Структура обработчика «удаление заказа»

На рисунках 1.9 представлена структура проекта.

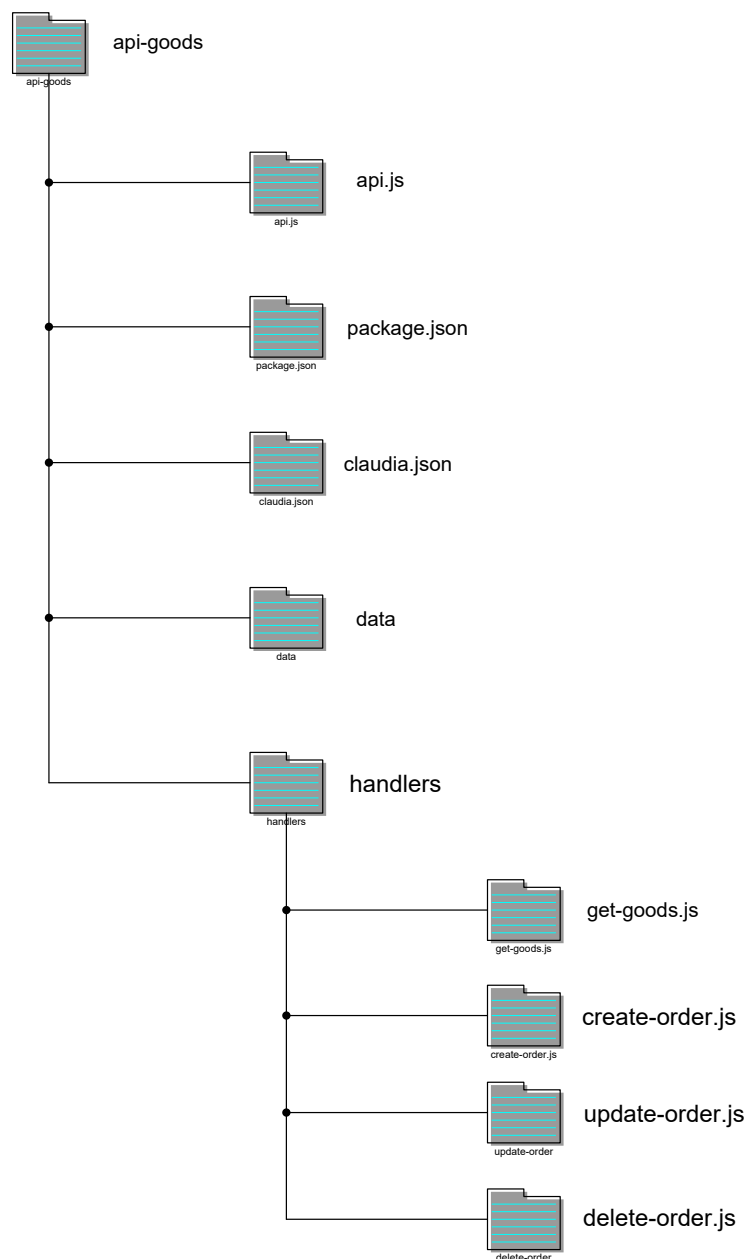


Рисунок 1.9 – Структура проекта

Отметим, что программист может и не знать, как расположен его код в Amazon Lambda функции. Для полноценной работы с проектом Node.js, достаточно классической структуры проекта, развернутого на локальной машине, доступа к аккаунту Amazon и установленного на компьютере

модуля Claudia API для автоматизированного размещения своих скриптов на Amazon.

База данных данного приложения, реализованная на базе Amazon DynamoDB и объектов данных на основе JSON.

Структура NoSQL СУБД выглядит следующим образом:

```
[
  {
    "id": 1,
    "name": "Nails",
    "size": [
      "small", "medium", "big"
    ]
  },
  {
    "id": 2,
    "name": "Screw",
    "size": [
      "small", "medium", "big"
    ]
  },
  {
    "id": 3,
    "name": "hammer",
    "type": [
      "metal", "wood", "common"
    ]
  },
  {
    "id": 4,
    "name": "drill",
    "type": [
      "electric", "hand"
    ]
  }
]
```

Она отлична от классической реляционной СУБД, но в данном проекте будет незаменима, т.к. позволяет быстро реализовать взаимодействие клиент сервер на базе JSON сообщений.

Ниже на рисунках 1.10 и 1.11 представлены основные потоки сообщений между слоями приложения.

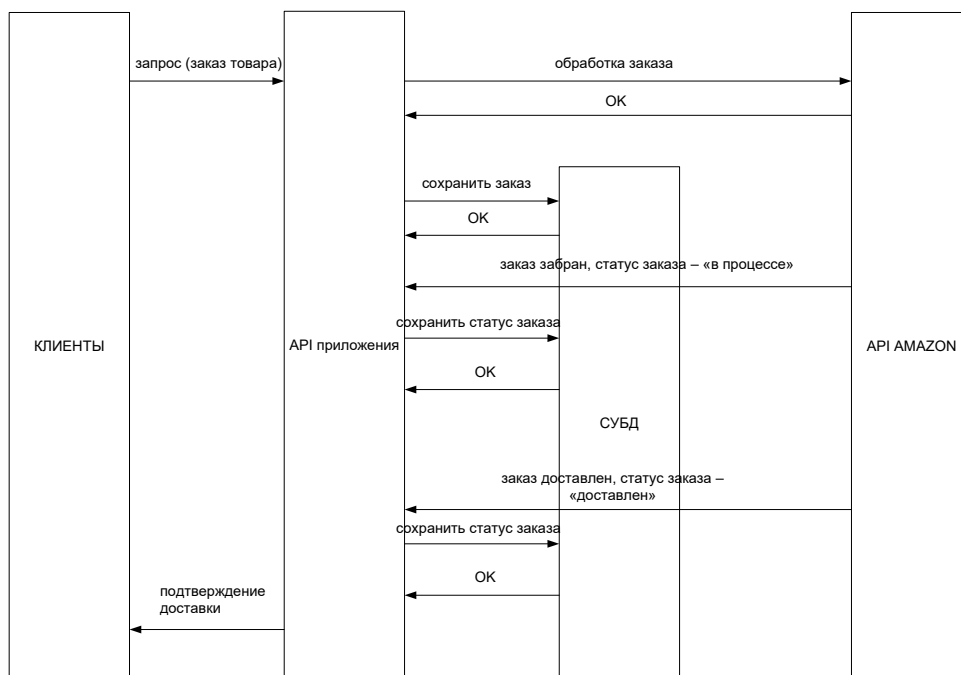


Рисунок 1.10 – Структура проекта

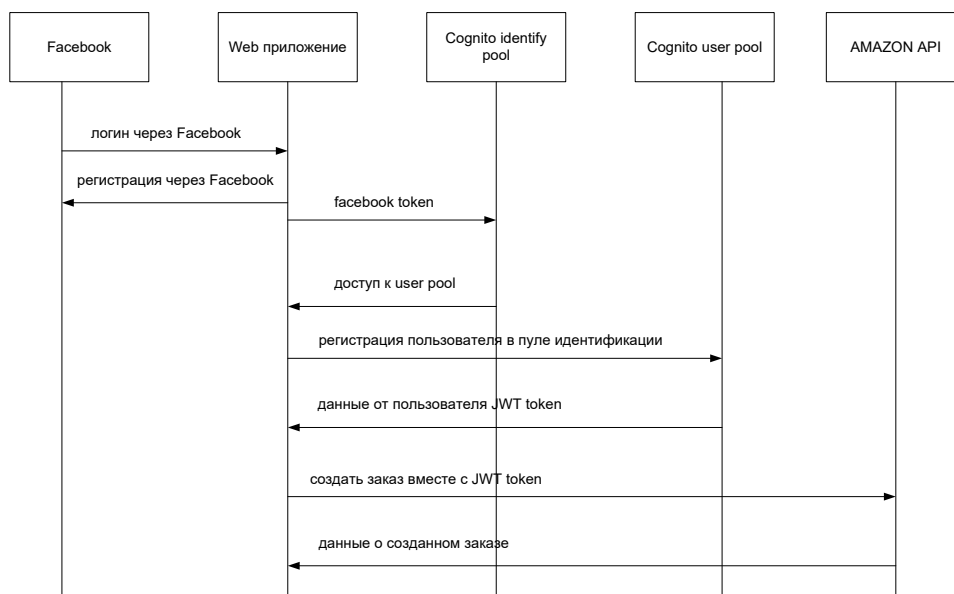


Рисунок 1.11 – Авторизация заказчиков через социальные сети

Механизмы регистрации пользователя, авторизации и аутентификации на базе платформы Amazon, реализуется с помощью сервиса Amazon Cognito.

Amazon Cognito позволяет добавлять возможности регистрации, авторизации и контроля доступа пользователей в мобильные и интернет-приложения. Фактически сервис позволяет интегрировать фактически готовые функции в приложение.

Amazon Cognito масштабируется до миллионов пользователей и поддерживает авторизацию с помощью социальных поставщиков удостоверений (Facebook, Google, Amazon).

В данном приложении планируется в качестве поставщика пользователей выбрать Facebook, т.к. API Facebook имеют дружественный интерфейс с сервисами Amazon.

1.4 Реализация механизмов работы с файлами

Работа с файлами в приложении будет реализован средствами API Amazon S3.

Amazon S3 (Amazon Simple Storage Service) позволяет хранить контент и получать к нему доступ из любого места в любое время.

Amazon S3 хранит данные в виде объектов в корзинах (bucket). Каждый объект представляет собой файл и, опционально метаданные, которые этот объект описывают (например медиа данные или картинки).

Чтобы сохранить данные в S3 - системе необходимо загрузить данные в корзину. После того, как данные загружены - можно установить ограничения доступа к нему.

Корзина - фактически это контейнер, который хранит объекты. Приложение может иметь неограниченное количество таких корзин, и для каждой можно установить доступ (кто может создавать, удалять и

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		25

просматривать список объектов в корзине), просматривать логи доступа к корзине и объектам в ней.

1.5 Реализация механизмов работы с файлами

В первой главе определена актуальность проекта, выбран стек технологий и определены следующие этапы разработки приложения, которые будут реализованы во второй главе проекта:

- создание и администрирование учетной записи AWS;
- настройка файла учетных данных AWS;
- установка Node.js и его менеджера пакетов, NPM;
- установка пакета автоматической загрузки приложения из среды разработки AWS. Настройка переменных среды, для глобальной видимости пакета обработчиков в разрабатываемой системе;
- тест разворачивания приложения в облачном пространстве AWS.

2 РАЗРАБОТКА ПРИЛОЖЕНИЯ

Разрабатываемая система представляет собой трехслойную архитектуру, в которой с точки зрения пользователя работает три участника (рисунок 2.1)

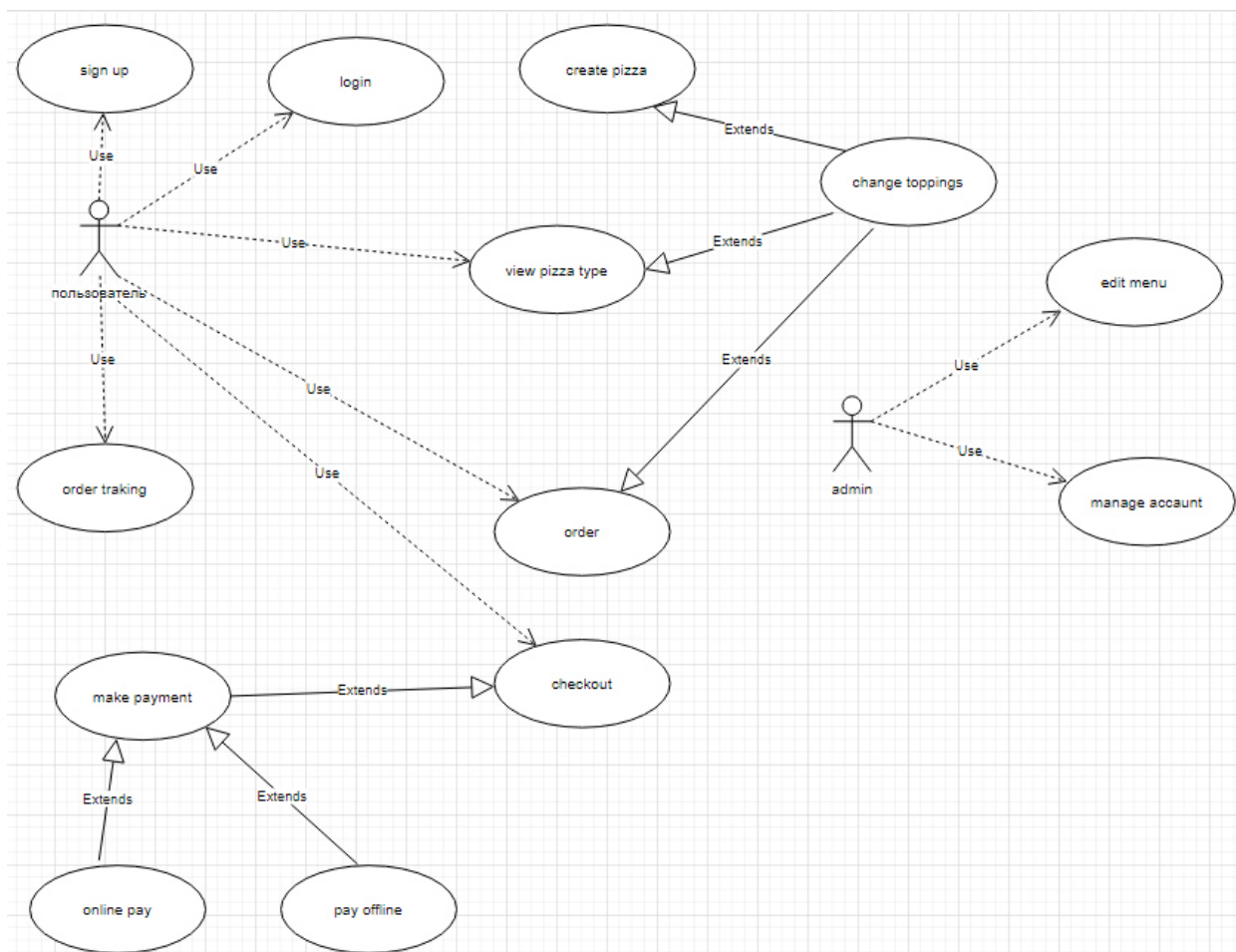


Рисунок 2.1 – Описание функций и участников системы.

Каждый из участников наделен определенным функционалом, который требуется реализовать в виде связки скрипт, база данных.

Так для пользователя данная связка выглядит следующим образом (рисунок 2.2).

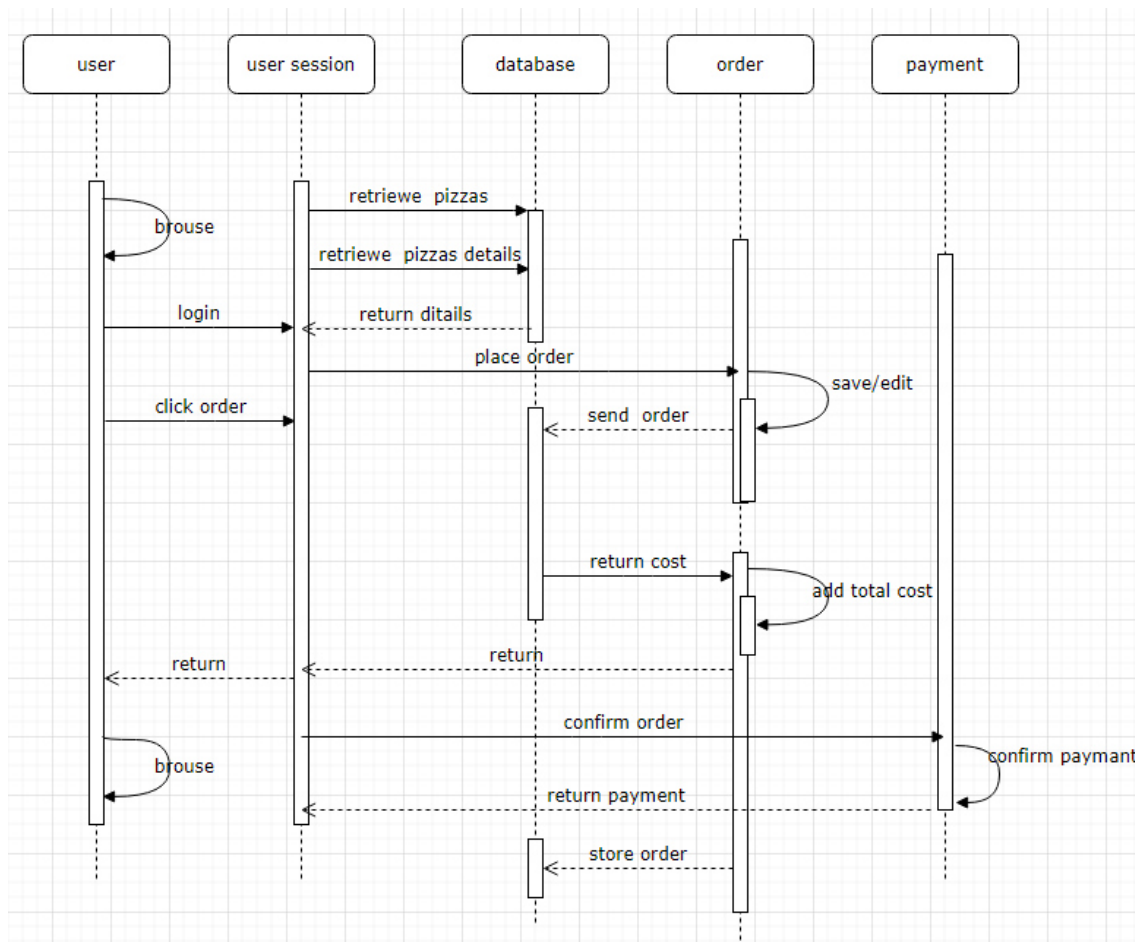


Рисунок 2.2 – Последовательность действий пользователя в системе

2.1 Разработка сервисов обработки GET и POST запросов приложения

Разработка обработчиков вывода списка всех товаров в галерее, получение заказа, создание заказа, обработка заказа, обновления статуса заказа, отмена заказа являются основой взаимодействия клиента и серверной части данного проекта, именно поэтому реализацию проекта начнем с сознания маршрутов взаимодействия клиента и сервера.

Каждый обработчик получает в качестве аргумента ссылку на функцию, содержащую программный код, для обработки маршрута и URL ссылку на данные и/или структуры данных для обработки логики.

Поскольку у приложения на данном этапе еще нет базы данных (СУБД показана в следующих разделах), в тестовых методах будет возвращаться простое текстовое сообщение в качестве ответа.

Формирование данных для вывода на главную страницу клиентской части приложения (информация о товаре), необходимо в корневом каталоге проекта создать файл JSON, в данном случае это будет файл pizzas.json

```
[
  {
    "id": 1,
    "name": "Capricciosa",
    "ingredients": [
      "tomato sauce", "mozzarella", "mushrooms", "ham",
      "olives"
    ]
  }, {
    "id": 2,
    "name": "Quattro Formaggi",
    "ingredients": [
      "tomato sauce", "mozzarella", "parmesan cheese", "blue
      cheese", "goat cheese"
    ]
  },
  {
    "id": 3,
    "name": "Napoletana",
    "ingredients": [
      "tomato sauce", "anchovies", "olives", "capers"
    ]
  }, {
    "id": 4,
    "name": "Margherita",
    "ingredients": [
      "tomato sauce", "mozzarella"
```

```

    ]
  }
]

```

В данный файл разработчик добавляет данные для работы системы, в последствии данные будут изменяться, добавляться, удаляться с помощью рабочего интерфейса программы, но без первоначальных данных проект не запустится на базе бессерверной площадки, т.к. она не имеет данных нужных системе и просто по умолчанию будет выдавать сообщение об ошибке.

Обработчик `getPizzas` формируется в отдельном файле. Все обработчики проекта требуется поместить в подкаталог `handlers` (создается в проектах Node.js по умолчанию). Код обработчика `getPizzas` будет храниться в отдельном файле `get-pizzas.js`.

В файле `get-pizzas.js` будет обработчик `getPizzas`, который возвращает список пицц из списка приведенного в файле `pizzas.json`. Сначала вам нужно импортировать список пиццы из созданного вами файла JSON. Во-вторых, вам нужно создать функцию обработчика `getPizzas` и экспортировать ее, чтобы вы могли запросить ее из входного файла. Затем, вместо того, чтобы просто возвращать список пиццы, сделайте еще один шаг и верните только одну пиццу, если идентификатор пиццы был передан в качестве параметра вашему обработчику `getPizzas`. Чтобы вернуть только одну пиццу, вы можете использовать строенный в пакет NodeJS метод `array.find`, который ищет товар по идентификатору пиццы из json списка. Если он найдет пиццу, то вернет ее `id` в качестве результата обработки. Если пиццы с таким идентификатором нет, метод выдаст ошибку `null`.

```

'use strict'

const pizzas = require('../data/pizzas.json')

function getPizzas(pizzaId) {
  if (!pizzaId)

```

```

        return pizzas

    const pizza = pizzas.find((pizza) => {
        return pizza.id == pizzaId
    })

    if (pizza)
        return pizza

    throw new Error('The pizza you requested was not found')
}

module.exports = getPizzas

```

После реализации обработчика получения товара по ID и списка всех товаров, необходимо реализовать оставшиеся маршруты и настроить точку входа нашего приложения. В NodeJS такой точкой входа по умолчанию является файл `api.js`.

```

'use strict'

const AWS = require('aws-sdk')
const docClient = new AWS.DynamoDB.DocumentClient()
const rp = require('minimal-request-promise')

const api = new Api()

const getPizzas = require('./handlers/get-pizzas')
const createOrder = require('./handlers/create-order')
const updateOrder = require('./handlers/update-order')
const deleteOrder = require('./handlers/delete-order')

api.get('/', () => 'index.html')

api.get('/pizzas', () => {

```

```

        return getPizzas()
    })
    api.get('/pizzas/{id}', (request) => {
        return getPizzas(request.pathParams.id)
    }, {
        error: 404
    })

    api.post('/orders', (request) => {
        return createOrder(request.body)
    }, {
        success: 201,
        error: 400
    })

    api.put('/orders/{id}', (request) => {
        return updateOrder(request.pathParams.id, request.body)
    }, {
        error: 400
    })

    api.delete('/orders/{id}', (request) => {
        return deleteOrder(request.pathParams.id)
    }, {
        error: 400
    })

    module.exports = api

```

Из листинга выше видно, что система работает следующим образом, при запуске приложения пользователь попадает по умолчанию на стартовую страницу index.html. Далее нажимая как вариант на элементы управления и/или ссылки, он инициирует работу конкретного маршрута. В данном случае функциональных маршрутов четыре, а точнее пять, так как маршрут

get позволяет получить выборку как одной конкретной единицы товара так и всего товарного ряда сразу.

По умолчанию API-шлюз (api.js) возвращает HTTP-статус 200 (201) для всех запросов. Синтаксис NodeJS позволяет помимо кодов об удачной работе маршрута отрабатывать и выдавать коды ошибок, в частности в нашем примере обрабатывается ошибка номер 404 (Сервер не найден). Данное свойство пригодится при обработке ошибок работы приложения, т.к. намного полезнее, если вместо стандартных ошибок пользователь сможет увидеть описание ошибки понятное для него а не только для программиста. В данном приложении код ошибки будет сигналом для роутера api.js о переводе пользователя на страницу «Ошибка. Перейдите по ссылке на главную страницу».

Следующая функция, которую требуется реализовать – это функция создания заказа. В качестве параметров данная функция принимает идентификатор товара и адрес заказчика. Если и тот и другой параметр проходят валидацию то функция выполняется, сохраняя заказ в СУБД, если данные заполнены не верно или отсутствуют, функция возвращает значение null.

```
'use strict'

const AWS = require('aws-sdk')
const docClient = new AWS.DynamoDB.DocumentClient()
const rp = require('minimal-request-promise')

const api = new Api()

function createOrder(request) {
  console.log('Save order', request.body)
  const userData = request.context.authorizer.claims
  console.log('User data', userData)
```

```

let userAddress = request.body && request.body.address
if (!userAddress) {
    userAddress = JSON.parse(userData.address).formatted
}

if (!request.body || !request.body.pizza || !userAddress)
    throw new Error('Не выбрана пицца и/или адрес доставки')

return rp.post(
    'https://fake-delivery-api.
    effortlessserverless.com/delivery', {
    headers: {
        Authorization: 'vladimirjur@-1234567890',
        'Content-type': 'application/json'
    },
    body: JSON.stringify({
        pickupTime: '15 min',
        pickupAddress: 'Penza, Mira 58',
        deliveryAddress: userAddress,
        webhookUrl: 'https://g8fhlgccof.execute-api.eu-
central-1.amazonaws.com/ vladimirjur/delivery',
    })
})

.then(rawResponse => JSON.parse(rawResponse.body))
.then(response => {
    return docClient.put({
        TableName: 'pizza-orders',
        Item: {
            cognitoUsername: userAddress['cognito:username'],
            orderId: response.deliveryId,
            pizza: request.body.pizza,
            address: userAddress,

```

```

        orderStatus: 'pending'
      }
    }).promise()
  })
  .then(res => {
    console.log('Order is saved!', res)

    return res
  })
  .catch(saveError => {
    console.log(`СУБД не сохранило заказ :( , saveError)

    throw saveError
  })
}

module.exports = createOrder

```

Как видно из листинга функции, в первую очередь подключается модуль Amazon AWS, который показывает, что данный код выполняется в бессерверном пространстве Amazon. Далее подключается база данных встроенная в эко систему AWS DynamoDB. Следующим шагом подготавливаются данные для заказа, заполняются переменные `userData`, `userAddress`. После проверки данных на наполненность идет подключение к аккаунту Amazon с контентом и функциями для отработки функции, в данном случае это путь `execute-api.eu-central-1.amazonaws.com/ vladimirjur`, ключом для подключения служит код авторизации `vladimirjur@-1234567890`, полученный разработчиком при создании аккаунта Amazon.

Сформированный заказ формируется в формате JSON и сохраняется с СУБД. При этом он будет виден как на странице заказов (в корзине), так и с СУБД до момента своего исполнения. После чего заказ будет удален.

2.2 Разработка обработчиков для СУБД

Amazon Web Services позволяет производить прототипирование веб-приложений, и реализовать маршрутный API в связке DynamoDB и API Gateway (как с Lambda функциями, так и без них) для настройки GET и POST запросов к базе с возможностью чтения, записи и изменения данных в ней.

Ниже представим механизм настройки Amazon DynamoDB и разработки базы данных для хранения информации о данных товара и заказов.

Прежде всего, необходимо зарегистрироваться и/или осуществить вход в окружение разработки AWS и затем войти в консоль. Создание сервиса СУБД начинается с настройки DynamoDB, для чего нажмите кнопку Create table, введите название таблицы apiData, основным ключ по которому в неё будут добавляться записи: userID и отметьте галочку Use default settings.

В DynamoDB строки в таблицу добавляются по указанному ключу, при этом возможно добавление любого количества параметров и не обязательно совпадение структуры данных для разных ключей. В нашем случае для каждого из пользователей по указанному ключу userID мы сможем добавлять любые данные, описывающие данного пользователя.

Далее необходимо создать так называемую роль в сервисе Identity and Access Management. В меню слева выберите раздел Roles и нажмите кнопку Create new role; укажите её название - dynamoAPI, и после нажатия кнопки Next Step - в разделе AWS Service Roles выберите Amazon API Gateway, ещё дважды нажмите Next Step и наконец - Create Role (рисунок 2.3).

Create DynamoDB table

[Tutorial](#)

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ⓘ

Primary key* Partition key

String ⓘ

☐ Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☒ Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

[Cancel](#)[Create](#)

Рисунок 2.3 – Создание таблицы СУБД

Для обработки заказов в формате приложение - СУБД необходимо сгенерировать ключ Role ARN в формате `arn:aws:iam::000000000000:role/roleName`.

Это значение необходимо будет использовать при связи запросов приложения с базой данных. Далее необходимо создать политику доступа для данного ключа, это делается на вкладке Permissions в разделе Inline policies, раскройте его и нажмите [click here](#) (рисунок 2.4).

В разделе Policy Generator нажмите кнопку Select, на открывшейся странице выберите сервис Amazon DynamoDB и укажите следующие Actions (действия которые приложение и его маршрутизаторы могут выполнять с СУБД):

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		37

- DeleteItem – удалять запись;
- GetItem – получать запись из БД;
- PutItem – записывать запись в БД;
- Query – делать запросы, в том числе и составные;
- Scan – сканировать БД;
- UpdateItem – обновлять состояние БД.

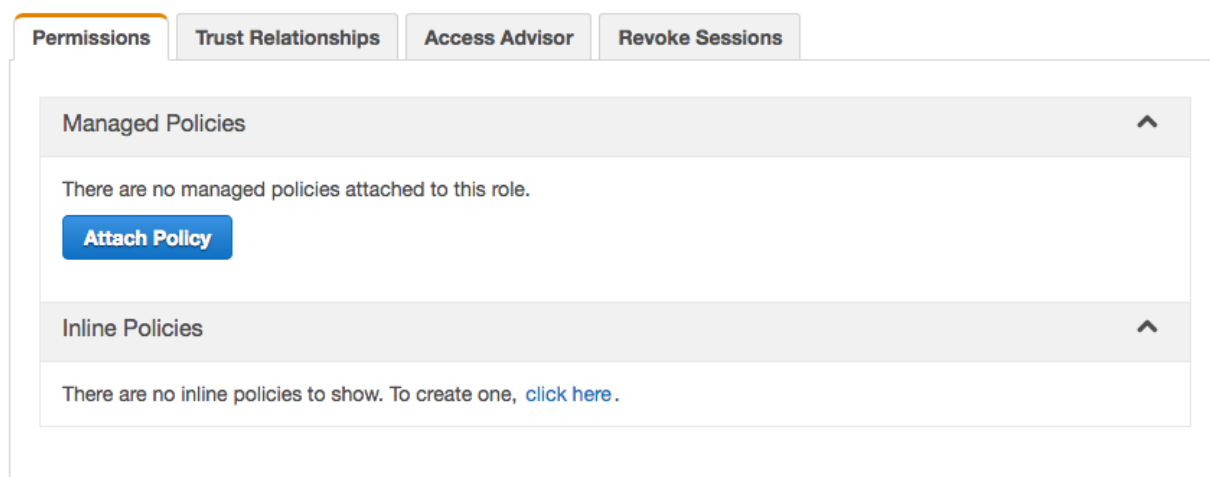


Рисунок 2.4 – Определение политики доступа приложения к СУБД

Если для реализации приложения требуется иметь возможность делать какие-то другие действия с помощью API, в таком случае требуется воспользоваться страницей справки Amazon DynamoDB. Что касается пункта Amazon Resource Name - здесь можно либо указать ARN таблицы (находится на вкладке Overview), либо указать *, это даст возможность пользователям с созданной ролью иметь доступ ко всем таблицам аккаунта программиста, иначе только авторизованный в Amazon пользователь может воспользоваться приложением. Нажмите кнопку Add Statement и Next Step, на открывшейся странице - Apply Policy. На этом настройка роли и первичная настройка СУБД завершены.

Переходим к созданию API с использованием сервиса API Gateway. Нажмите синюю кнопку Create API, на открывшейся страницу укажите его

название - The API и нажмите кнопку Create API внизу страницы. Теперь необходимо создать ресурс, к которому можно будет обращаться с запросами, нажмите кнопку Actions и выберите Create Resource (рисунок 2.5).

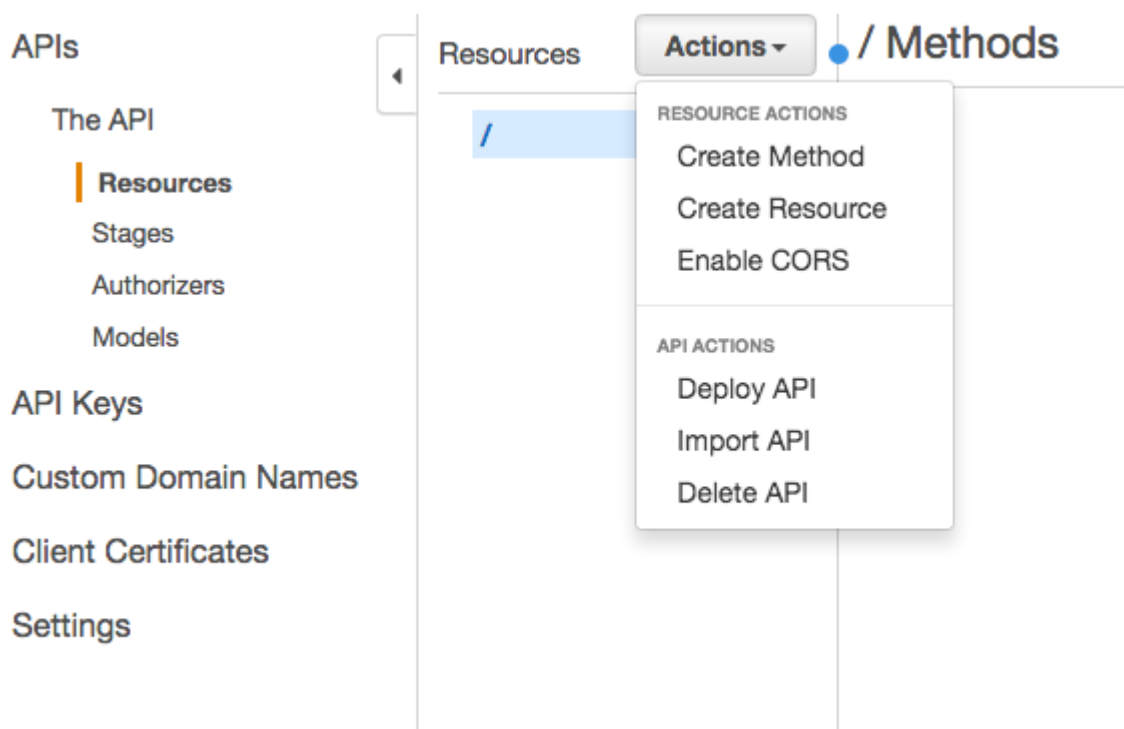


Рисунок 2.5 – Связывание программного обработчика и СУБД

Назовём данный ресурс pizzas, он будет содержать информацию о товаре, и чтобы получить доступ к конкретному товару необходимо будет указать ID товара в качестве параметра пути. Чтобы в сервисе API Gateway создавать такие параметры, необходимо создать новый ресурс на уровень ниже уже созданного pizzas, и в качестве Resource Name и Resource Path указать {pizzasid}. При указании названия в таком формате Resource Path автоматом заменяется на -pizzasid-, разработчику необходимо вручную указать нужную форму для параметра пути (рисунок 2.6).

Resources Actions ▾

▼ / /pizzas

New Child Resource

Use this page to create a new child resource for your resource. ●

Resource Name*

Resource Path*

You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'.

* Required

Cancel Create Resource

Рисунок 2.6 – Формирование полей таблицы.

Далее создадим метод для создания записи о новом товаре, для этого выберем ресурс {pizzaid}, и нажав кнопку Actions выберем Create method, в нем укажем его тип – POST и нажмем галочку для его создания. В открывшемся меню настроек в разделе Integration type необходимо открыть спойлер Show advanced и выбрать AWS Service Proxy. Настройки выбранные для реализуемого приложения(рисунок 2.7):

- AWS Region укажите регион, на серверах которого будет храниться база данных (по умолчанию - us-east-1, проверить его можно в разделе Overview созданной таблицы);
- AWS Service: DynamoDB;
- AWS Subdomain: оставить пустым;
- HTTP method: POST (используется для любых обращений к DynamoDB, в том числе и для получения данных);
- Action Type: Use action name;
- Action: PutItem (используется для создания новой записи/перезаписи всего значения по указанному ключу);

- Execution role: указать роль, которую создали, в формате `arn:aws:iam::000000000000:role/roleName`.

Integration type ☐ Lambda Function
☐ HTTP Proxy
☐ Mock Integration
☒ AWS Service Proxy

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action Type ☒ Use action name
☐ Use path override

Action

Execution role ⓘ

Рисунок 2.7 – Связь обработчиков и БД

После сохранения этих настроек необходимо сперва зайти в первый квадрат Method Request, в пункте API Key Required выбрать true и нажать на галочку чтобы сохранить настройки это необходимо для доступа к данному методу извне с помощью токена авторизации. Вернитесь назад и зайдите во второй квадрат Integration Request для настройки собственно запроса в DynamoDB. На открывшейся странице промотайте в самый низ и откройте раздел Body Mapping Templates, нажмите кнопку Add mapping template,

укажите Content type: application/json, в поле ввода необходимо указать параметры запроса, для создания новой записи используйте следующий код:

```
{
  "users": [
    {
      "email": "vladimiryi@gmail.com.com",
      "password": "123456",
      "confirmPassword": "123456",
      "phone": "+79273876549",
      "id": 1
    },
    {
      "email": "admin",
      "password": "123",
      "confirmPassword": "123",
      "phone": "123",
      "id": 2
    },
    {
      "email": "admin@qq.com",
      "password": "123",
      "confirmPassword": "123",
      "phone": "123",
      "id": 3
    }
  ],
  "addPizza": [
    {
      "name": " Margherita",
      "description": " tomato sauce", "mozzarella",
      "options": [
        {
          "size": "9",
          "price": "38"
        },
        {
          "size": "12",
          "price": "45"
        }
      ],
      "id": 1
    },
    {
      "name": " Capricciosa",
```

```

        "description": "tomato sauce", "mozzarella",
        "mushrooms", "ham", "olives",
        "options": [
            {
                "size": "9",
                "price": "34"
            },
            {
                "size": "12",
                "price": "45"
            }
        ],
        "id": 2
    },
    {
        "name": " Quattro Formaggi",
        "description": "tomato sauce", "mozzarella",
        "parmesan cheese", "blue cheese", "goat cheese",
        "options": [
            {
                "size": "12",
                "price": "56"
            },
            {
                "size": "24",
                "price": "106"
            }
        ],
        "id": 3
    }
]
}

```

Здесь указывается имя таблицы, в которой производим изменения, первым указывается ключ, по которому будут вноситься данные: pizzaID, его значение берётся из параметра пути pizzaid. Из тела запроса берутся данные по ключу parameter и добавляются в одноимённый столбец базы данных для указанного товара. В качестве ответа присылается предыдущее значение по указанному ключу, если оно существовало. Если пицца с таким именем не существует, придёт пустой ответ.

Проверить работоспособность запроса можно с помощью диалога, для этого вверху страницы нажмите кнопку назад и справа от четырёх квадратов — нажмите на кнопку Test со значком «молния» (рисунок 2.8).

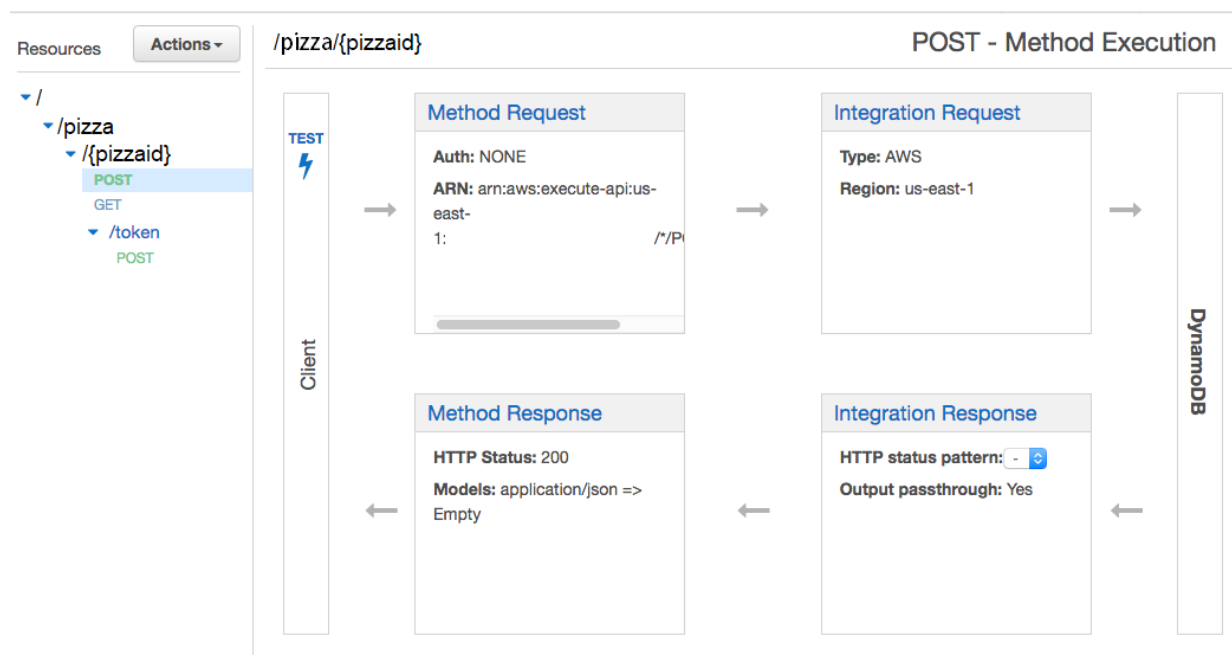


Рисунок 2.8 – Исполнение запроса

В открывшемся окне нужно будет указать значение Path parameter - это имя заказа (заказанной пиццы), которого мы хотим создать/пересоздать (напомню, PutItem перезаписывает всю строку по указанному ключу), чуть ниже указываем тело (body) запроса (рисунок 2.9):

```
{ "parameter" ID: "11" }
```

Resources
Actions

/
/pizza
/{pizzaId}
POST
GET
/token
POST

Method Execution
/user/{userid} - POST - Method Test

Make a test call to your method with the provided input

Path
userid

getPizzas

Request: /user/newUserOne
Status: 200
Latency: 118 ms
Response Body

{}

Query Strings
No query string parameters exist for this method. You can add them via Method Request.

Response Headers

{"Content-Type":"application/json"}

Headers
No header parameters exist for this method. You can add them via Method Request.

Logs

Stage Variables
No stage variables exist for this method.

Execution log for request test-request
Wed Aug 03 21:04:27 UTC 2016 : Starting execution for request: test-invoke-request
Wed Aug 03 21:04:27 UTC 2016 : HTTP Method: POST, Resource Path: /user/{userid}
Wed Aug 03 21:04:27 UTC 2016 : Method request path: {userid=newUserOne}
Wed Aug 03 21:04:27 UTC 2016 : Method request query string: {}
Wed Aug 03 21:04:27 UTC 2016 : Method request headers: {}
Wed Aug 03 21:04:27 UTC 2016 : Method request body before transformations: {
"parameter": "112233"
}

Client Certificate
No client certificates have been generated.

Request Body

Рисунок 2.9 – Результат работы с БД

Для чтения данных о заказе необходимо так же в рамках ресурса {userid} создать GET метод с Action Query, произвести настройку интеграционного запроса (в этом случае запрос к базе делается POST методом) и указать следующий шаблон:

```

'use strict'

const AWS = require('aws-sdk')

const docClient = new AWS.DynamoDB.DocumentClient()

function getOrders(orderId) {

  if (typeof orderId === 'undefined')

    return docClient.scan({

      TableName: 'pizza-orders'

    }).promise()

    .then(result => result.Items)

```

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		45

```

return docClient.get({
  TableName: 'pizza-orders',
  Key: {
    orderId: orderId
  }
}).promise()
  .then(result => result.Item)
}

module.exports = getOrders

```

Если же хотим изменить какое-то значения параметра для определённого заказа, не меняя все остальные заказы, то также можем использовать POST метод с Action типа UpdateItem и следующим шаблоном:

```

'use strict'

const AWS = require('aws-sdk')
const docClient = new AWS.DynamoDB.DocumentClient()
function updateOrder(orderId, options) {
  if (!options || !options.pizza || !options.address)
    throw new Error('Данные не корректны')
  return docClient.update({
    TableName: 'pizza-orders',
    Key: { // <3>
      orderId: orderId
    },
    UpdateExpression: 'set pizza = :p, address=:a',
    ExpressionAttributeValues: {
      ':p': options.pizza,
      ':a': options.address
    },
    ReturnValues: 'ALL_NEW'
  })
}

```

```

    }).promise()
    .then((result) => {
        console.log('Order is updated!', result)
        return result.Attributes
    })
    .catch((updateError) => {
        console.log(`Oops, order is not updated :(`,
updateError)
        throw updateError
    })
}
module.exports = updateOrder

```

В этом случае запрос возвращает все обновлённые данные о заказе в случае успеха.

Для того чтобы схема работы база данных – программная среда работали и за пределами экосистемы Amazon, требуется реализовать следующие настройки:

Для настройки доступа к Amazon API и Amazon DB извне, для этого необходимо нажать кнопку Actions в меню Amazon API и выбрать Deploy API. Указываете следующие параметры:

- Deployment stage: [New Stage];
- Stage name: apiRelease.

Нажимаете Deploy и получаете Invoke URL следующего вида: m000a.execute-api.us-east-1.amazonaws.com/apiPizza.

Чтобы делать запросы по этому ключу, необходим токен авторизации (в нашем случае aws-sdk), для его получения в меню слева зайдите в раздел API Keys, нажмите кнопку Create, назовите как-то свой ключ и сохраните его. Далее в появившемся разделе API Stage Association выберите нужный API и недавно созданную сцену, после нажмите кнопку Add. Вернитесь через левое

меню в наш API, выберите Actions->Deploy API, выберите уже созданную Stage и нажмите Deploy.

Теперь можно делать запросы к Amazon API извне, по ключу, который требуется указать в скриптах программного кода нашего фронтенд приложения.

2.3 Разработка СУБД

СУБД Amazon DynamoDB является представителем NoSQL баз данных.

В DynamoDB запись хранится как документ JSON. Каждая таблица, или элемент хранятся в качестве атрибутов в едином документе. В такой модели данные оптимизированы для интуитивно понятной разработки и горизонтальной масштабируемости.

Но для наглядности представим данные в реляционном виде, в виде таблиц (таблица 2.1-2.11).

Таблица 2.1 – customer – содержит информацию о заказчике товара

№	наименование	тип
1	id	SERIAL NOT NULL
2	name	VARCHAR(60)
3	address	VARCHAR(60)
4	phone	VARCHAR(14)
5	PRIMARY KEY (id)	

Таблица 2.2 – credit_card – содержит информацию о заказчике товара

№	наименование	тип
1	2	3
1	id	SERIAL NOT NULL
2	name	VARCHAR(60)

Продолжение таблица 2.2

1	2	3
3	number	VARCHAR(20)
4	phone	VARCHAR(14)
5	expiration	DATE
6	UNIQUE ("name", "number")	
7	PRIMARY KEY (id)	

Таблица 2.3 – customer_credit_card –таблица связка пользователя и кредитную карту

№	наименование	тип
1	customer_id	INTEGER
2	credit_card_id	INTEGER

Таблица 2.4 – customer_order – таблица связка пользователя и кредитную карту

№	наименование	тип
1	id	SERIAL NOT NULL
2	customer_id	INTEGER
3	credit_card_id	INTEGER
4	total_price	MONEY
5	PRIMARY KEY ("id")	

Таблица 2.5 – customer_order_preference – таблица связка пользователя и заказа

№	наименование	тип
1	customer_id	INTEGER
2	customer_order_id	INTEGER

Таблица 2.6 – drinks – таблица содержит список напитков, которые пользователь может заказать

№	наименование	тип
1	Name	VARCHAR(20)
2	Price	MONEY
3	PRIMARY KEY ("name")	

Таблица 2.7 – pizzas – таблица содержит список сортов пиццы, которые пользователь может заказать

№	наименование	тип
1	id	SERIAL
2	size	VARCHAR(2)
3	topping_1	VARCHAR(20)
4	topping_2	VARCHAR(20)

Таблица 2.8 – customer_pizzas – таблица связка пользователя и заказа

№	наименование	тип
1	customer_order_id	INTEGER
2	pizza_id	INTEGER

Таблица 2.9 – toppings – таблица связка пользователя и заказа

№	наименование	тип
1	Name	VARCHAR(20)
2	PRIMARY KEY ("name")	

Таблица 2.10 – customer_drinks – таблица связка пользователя и заказа

№	наименование	тип
1	customer_order_id	INTEGER
2	drink	VARCHAR(20)

Таблица 2.11 – sizes – таблица содержит список напитков, которые пользователь может заказать

№	наименование	тип
1	name	VARCHAR(2)
2	Price	MONEY
3	PRIMARY KEY ("name")	

На основании таблиц 2.1- 2.11 реализуется схематехническое описание БД (рисунок 2.10).

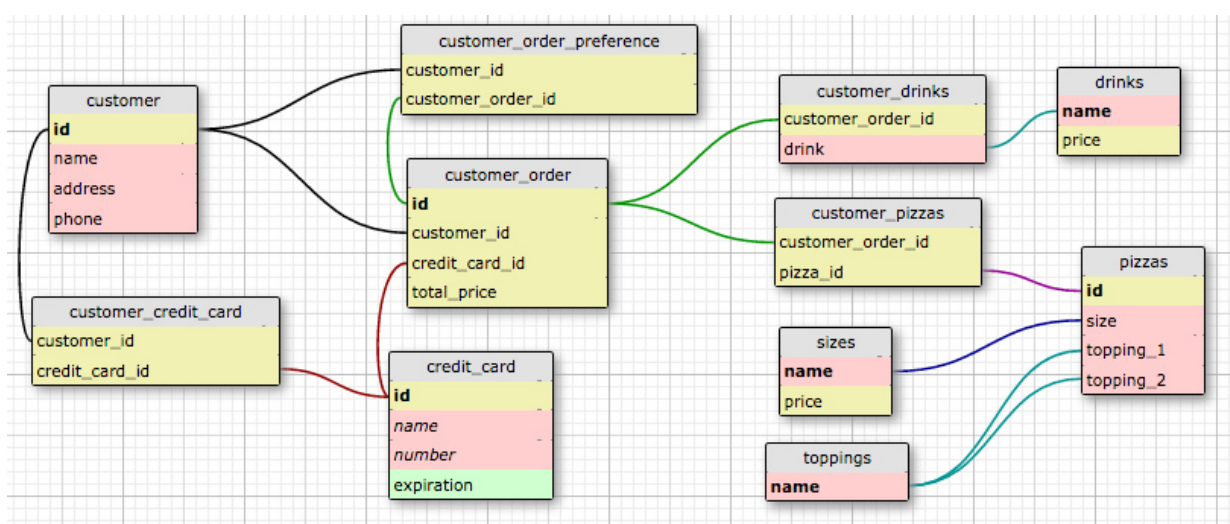


Рисунок 2.10 – Схема БД приложения

NoSQL описание БД, используемое в среде Amazon представлено в приложении А.

2.4 Интерфейсы приложения

В данном разделе описана работа приложения с точки зрения пользователей, представлены интерфейсы и результат реализации обработчиков и СУБД, описанных выше.

Пользователь при заходе на сайт попадает на главную страницу приложения, которая фактически отрабатывает маршрут `getPizzas()`, показывая весь ассортимент товара, начиная с новинок ассортимента (рисунок 2.11).

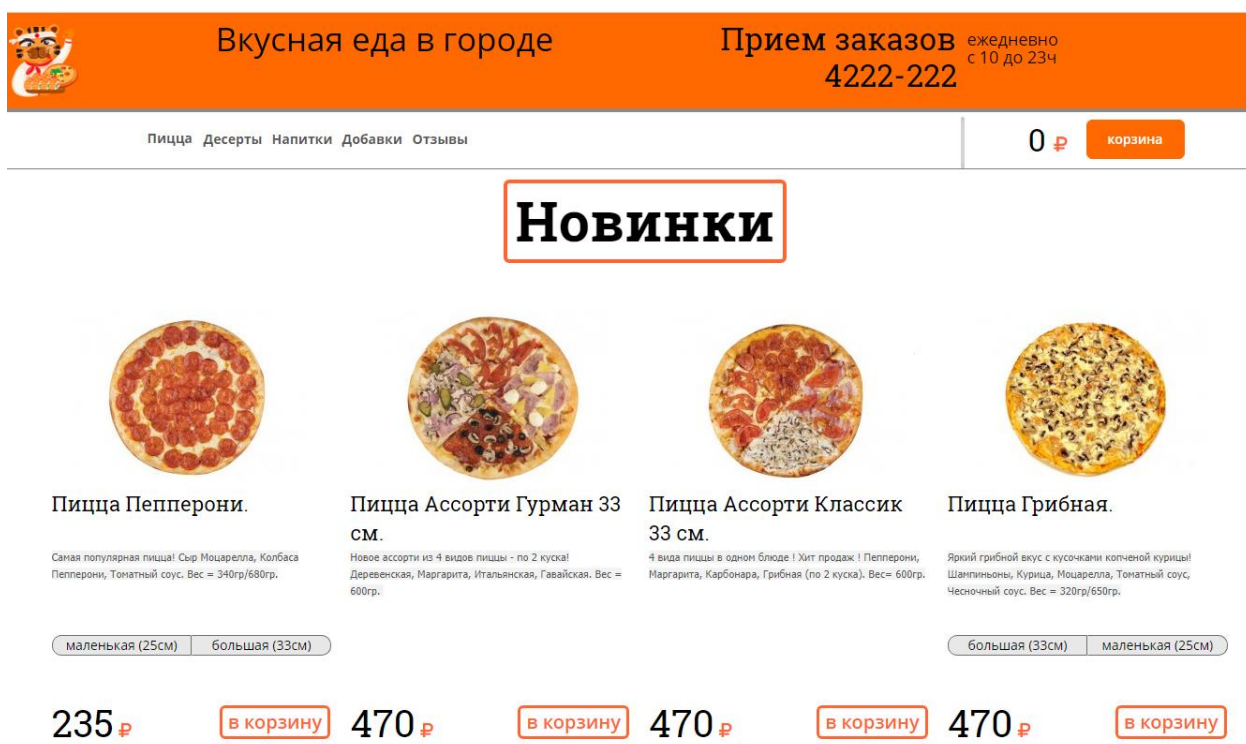


Рисунок 2.11 – Главная страница приложения

Элементы страницы «Новинки» показаны в таблице 2.12.

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		52

Таблица 2.12 - Элементы главной страницы

Обозначение элемента	Компонент	Описание
1	href	Ссылки на другие страницы сайта
2	table	Информация о товаре
3	button	Выбор размера пиццы
4	button	Корзина

Далее пользователь благодаря навигационному меню в верхней части сайта может осуществить переход на страницу «Пицца» (рисунок 2.12), по своей структуре данная страница совпадает со страницей «Новинки» и содержит такие же элементы что и она (таблица 2.12).



Вкусная еда в городе

Прием заказов ежедневно с 10 до 23ч
4222-222

Пицца Десерты Напитки Добавки Отзывы

0  

Пицца



Пицца Пепперони.
Самая популярная пицца! Сыр Моцарелла, Колбаса Пепперони, Томатный соус. Вес = 340гр/680гр.

маленькая (25см)

большая (33см)

235 

 470 



Пицца Ассорти Гурман 33 см.
Новое ассорти из 4 видов пиццы - по 2 куса! Деревенская, Маргарита, Итальянская, Гавайская. Вес = 600гр.

большая (33см)

маленькая (25см)

470 

 470 



Пицца Ассорти Классик 33 см.
4 вида пиццы в одном блюде ! Хит продаж ! Пепперони, Маргарита, Карбонара, Грибная (по 2 куса). Вес= 600гр.

большая (33см)

маленькая (25см)

470 

 470 



Пицца Грибная.
Яркий грибной вкус с кусочками копченой курицы! Шампиньоны, Курица, Моцарелла, Томатный соус, Чесночный соус. Вес = 320гр/650гр.

большая (33см)

маленькая (25см)

470 

 470 

Рисунок 2.12 – Главная страница «Пицца»

Выбор конкретного экземпляра товара, приводит к переходу на страницу с характеристиками товара: ценой и размерами (рисунок 2.13 и 2.14). Элементы страницы выбора товара показаны в таблице 2.13.

Таблица 2.13 - Элементы страницы выбранного товара

Обозначение элемента	Компонент	Описание
1	href	Ссылки на другие страницы сайта
2	table	Информация о товаре
3	button	Выбор размера пиццы
4	button	Корзина

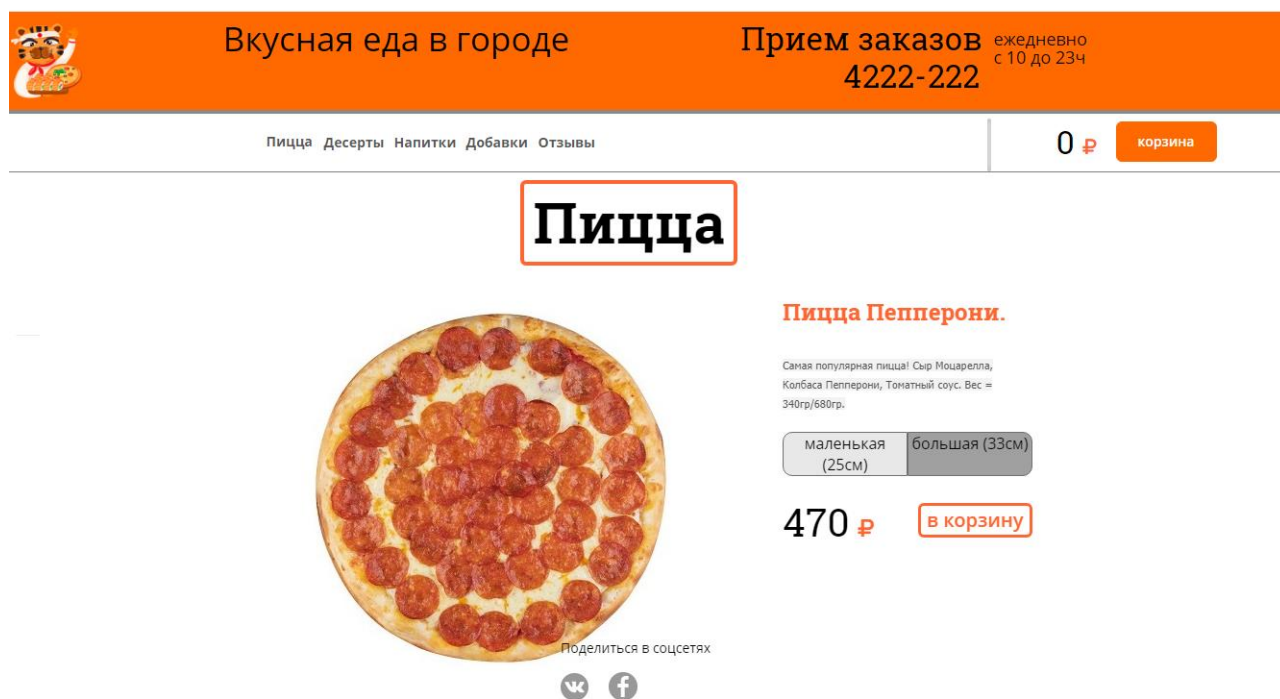
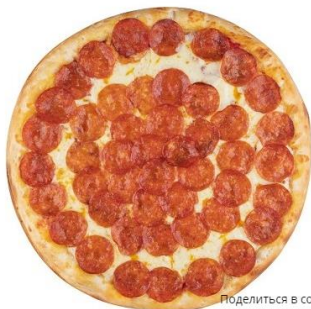


Рисунок 2.13 – Страница выбранного товара

Пицца



Пицца Пепперони.

Самая популярная пицца! Сыр Моцарелла, Колбаса Пепперони, Томатный соус. Вес = 340гр/680гр.

маленькая (25см)

большая (33см)

235 ₽

в корзину

Поделиться в соцсетях



Рисунок 2.14 – Изменение характеристик товара

Нажатие кнопки «в корзину», возвращает пользователя на страницу «Пицца», в счетчике пиццы появляется количество заказанного товара, которое можно регулировать (рисунок 2.15).



Вкусная еда в городе
Прием заказов 4222-222

ежедневно с 10 до 23ч

Пицца
Десерты
Напитки
Добавки
Отзывы

235 ₽

корзина

Пицца



♥

Пицца Пепперони.

Самая популярная пицца! Сыр Моцарелла, Колбаса Пепперони, Томатный соус. Вес = 340гр/680гр.

маленькая (25см)

большая (33см)

235 ₽

▼

1

▲

470 ₽

в корзину



♥

Пицца Ассорти Гурман 33 см.

Новое ассорти из 4 видов пиццы - по 2 кусочка! Деревенская, Маргарита, Итальянская, Гавайская. Вес = 600гр.

большая (33см)

470 ₽

в корзину



♥

Пицца Ассорти Классик 33 см.

4 вида пиццы в одном блюде! Хит продаж! Пепперони, Маргарита, Карбонара, Грибная (по 2 кусочка). Вес= 600гр.

большая (33см)

470 ₽

в корзину



♥

Пицца Грибная.

Яркий грибной вкус с кусочками копченой курицы! Шампиньоны, Курица, Моцарелла, Томатный соус, Чесночный соус. Вес = 320гр/650гр.

большая (33см)

маленькая (25см)

470 ₽

в корзину

Рисунок 2.15 – Выбранный товар

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		55

Таблица 2.14 – Элементы страницы «Пицца» с выбранным товаром

Обозначение элемента	Компонент	Описание
1	href	Ссылки на другие страницы сайта
2	table	Информация о товаре
3	button	Выбор размера пиццы
4	button	Корзина
5	javascript	Представление и изменение диапазона числа выбранного товара

При работе со страницами выбора напитков и другой продукции происходит реализация таких же конструкций и шаблонов, которые описаны выше (рисунок 2.16 и 2.17). Страница состоит из элементов, представленных в таблице 2.15.

Таблица 2.15 - Элементы страницы «Напитки» с выбранным товаром

Обозначение элемента	Компонент	Описание
1	href	Ссылки на другие страницы сайта
2	table	Информация о товаре
3	button	Выбор размера пиццы
4	button	Корзина
5	javascript	Представление и изменение диапазона числа выбранного товара

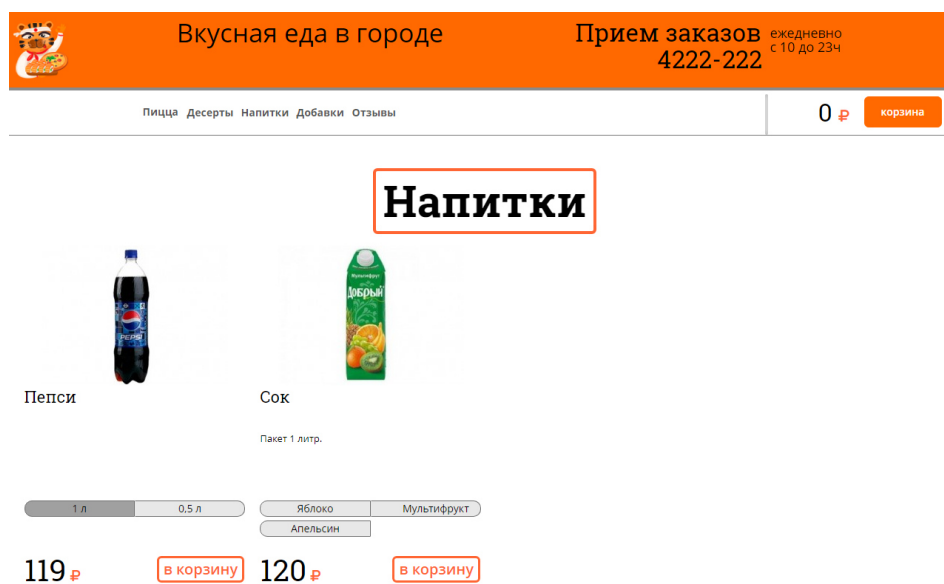


Рисунок 2.16 – Страница Выбор напитков, без выбора товара

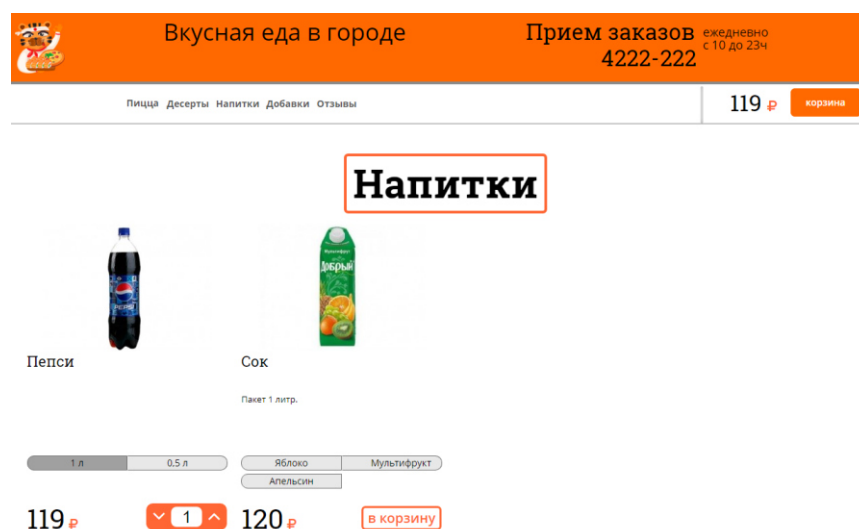


Рисунок 2.17 – Страница Выбор напитков, с выбранным товаром

Пользователь оформивший заказ, переходит на страницу «Корзина» (рисунок 2.18 и 2.19). Страница состоит из элементов, представленных в таблице 2.16.

Таблица 2.16 – Элементы страницы «Оформление заказа» с выбранным товаром

Обозначение элемента	Компонент	Описание
1	href	Ссылки на другие страницы сайта
2	Table	Информация о товаре первого типа
3	Table	Информация о товаре второго типа
4	javascript	Представление и изменение диапазона числа выбранного товара первого типа
5	javascript	Представление и изменение диапазона числа выбранного товара второго типа



Вкусная еда в городе
Прием заказов ежедневно с 10 до 23ч
4222-222

[Пицца](#)
[Десерты](#)
[Напитки](#)
[Добавки](#)
[Отзывы](#)

334 ₽

корзина

Корзина > Оформление заказа > Заказ принят >

Корзина



Пицца Пепперони. маленькая (25см)

Самая популярная пицца! Сыр Моцарелла, Колбаса Пепперони, Томатный соус. Вес = 340гр/680гр.

235 ₽

▼

1

▲

✕



Пепси

0,5 л

99 ₽

▼

1

▲

✕

Наценка за доставку от 100₽

Сумма заказа: 434₽

Назад в меню

оформить

Рисунок 2.18 – Страница просмотра содержимого корзины



Вкусная еда в городе
Прием заказов
4222-222

ежедневно
с 10 до 23ч

[Пицца](#)
[Десерты](#)
[Напитки](#)
[Добавки](#)
[Отзывы](#)

235 ₽

корзина

Корзина > Оформление заказа > Заказ принят >

Корзина



Пицца Пепперони.
маленькая (25см)
Самая популярная пицца! Сыр Моцарелла, Колбаса Пепперони, Томатный соус. Вес = 340гр/680гр.

235 ₽

▼

1

▲

Назад в меню

оформить

Рисунок 2.19 – Редактирование содержимого корзины

В конечном итоге пользователь оформляет заказ, для чего на странице «Оформление заказа» (рисунок 2.20). Страница состоит из элементов, представленных в таблице 2.17.

Таблица 2.17 – Элементы страницы «Оформление заказа» с выбранным товаром

Обозначение элемента	Компонент	Описание
1	href	Ссылки на другие страницы сайта
2	Table	Информация о заказе
3	input	Контактные данные
4	input	Адрес доставки
5	input type="radio"	Выбор способа оплаты
6	button	Кнопка подтверждения

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		59



Вкусная еда в городе
Прием заказов
4222-222

ежедневно
с 10 до 23ч

235 ₽

корзина

Корзина > Оформление заказа > Заказ принят >

доставка

самовывоз

Контактные данные

Имя*

Телефон*

Адрес доставки

Улица*

Дом*

Корпус/стр

Подъезд

Этаж

Кв

Выбор оплаты

☐ Банковской картой курьеру
☒ Наличными курьеру

Сумма с которой готовить сдачу

подтвердить

Ваш заказ

Пицца Пепперони.
маленькая (25см)

1 x 235 ₽

Наценка за доставку от 100₽

Итого 335₽

Рисунок 2.20 – Страница оформления заказа

На основании выше описанного интерфейса, показана работа пользователя на уровне web представления приложения.

Пользователь с правами администратора системы имеет возможность моделировать продукцию: добавлять новые позиции, удалять старые и/или изменять имеющиеся характеристики товара (рисунки 2.21-2.25).

Ниже представлены основные шаги администратора.

Попадая на главное меню, администратор может добавлять товар и разделы. В данном примере представлены иллюстрации работы с товаром.

На рисунке 2.22 представлена кнопка «Добавить пиццу», по нажатию на которую пользователь попадает на страницу «Добавить», на которой пользователь формирует новую пиццу, сформированную пиццу можно сохранить и изменить ее параметры, если это требуется.

На главной странице можно также нажатием на кнопку «Корзина» удалить любую позицию.

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		60

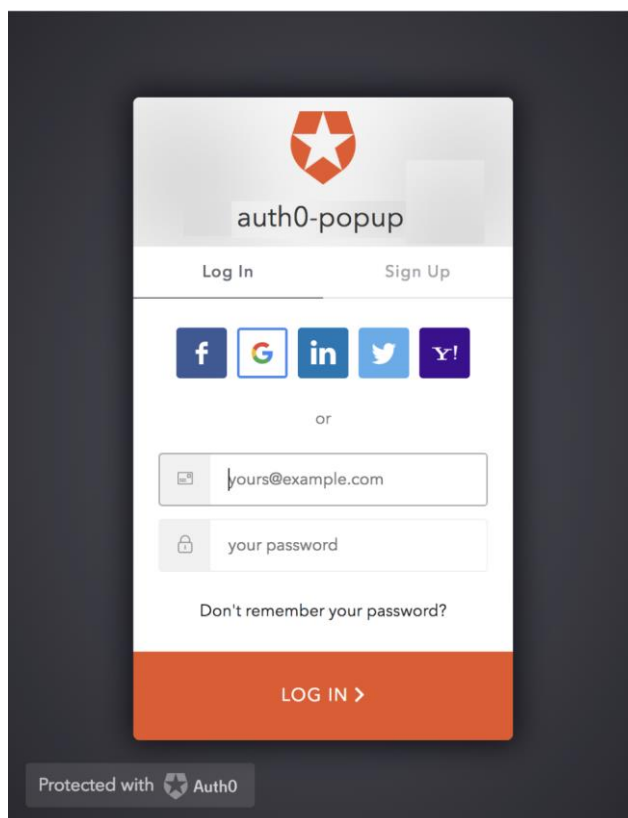


Рисунок 2.21 – Вход в систему с правами администратора

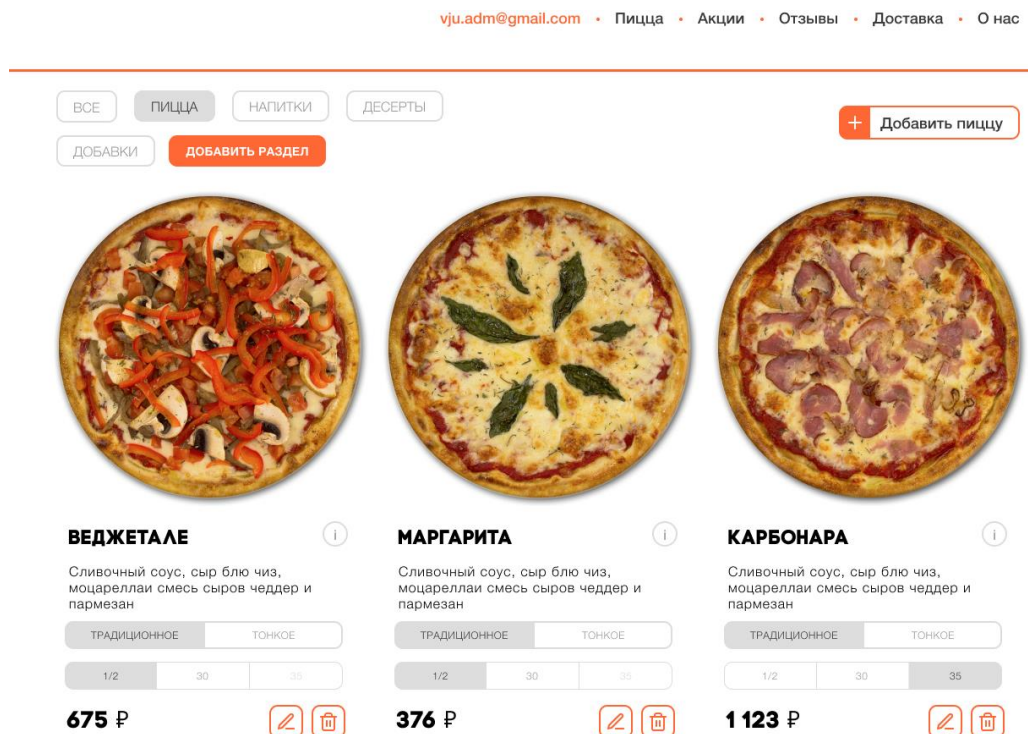


Рисунок 2.22– Основное меню работы с товарами

↑ ЗАГРУЗИТЬ ИЗОБРАЖЕНИЕ

ДОБАВИТЬ НАЗВАНИЕ

✓ ДОБАВИТЬ ИНГРИДИЕНТ

Добавить описание

ПИЩЕВАЯ ЦЕННОСТЬ НА 100 Г

МАССА НЕТТО

1/2

30

35

БОЛЬШЕ СЫРА

ОСТРЕЕ

ЗАДАТЬ ЦЕНУ В Р

СОХРАНИТЬ

Рисунок 2.23 – Добавить товар

vju.adm@gmail.com • Пицца • Акции • Отзывы • Доставка • О нас

ВЕДЖЕТАЛЕ

✓ С СЫРОМ

✓ С ГРИБАМИ

✓ ДОБАВИТЬ ИНГРИДИЕНТ

Сливочный соус, сыр блю чиз, моцареллаи смесь сыров чеддер и пармезан

ПИЩЕВАЯ ЦЕННОСТЬ НА 100 Г

МАССА НЕТТО

1/2

30

+

БОЛЬШЕ СЫРА

ОСТРЕЕ

675 Р

СОХРАНИТЬ

Рисунок 2.24 – Редактировать товар

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		62



ВЕДЖЕТАЛЕ

- ✓ С СЫРОМ
- ✓ С ГРИБАМИ

Сливочный соус, сыр блю чиз, моцареллаи смесь сыров чеддер и пармезан

ПИЩЕВАЯ ЦЕННОСТЬ НА 100 Г
МАССА НЕТТО

188,6 ККАЛ
470±50 Г

1/2

30

- БОЛЬШЕ СЫРА
- ОСТРЕЕ

675 Р

ИЗМЕНИТЬ

Рисунок 2.25 – Редактировать товар

3 ТЕХНОЛОГИЯ РАЗРАБОТКИ ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ

В технологическом разделе данной работы опишем способ связывания нашего web интерфейса со скриптами и базой данных, расположенными в облачной инфраструктуре Amazon.

Графический интерфейс реализуется с использованием HTML, CSS и JavaScript. В качестве среды разработки выбрана IDE NetBeans с настроенным входом в аккаунт AWS Amazon (рисунок 3.1).

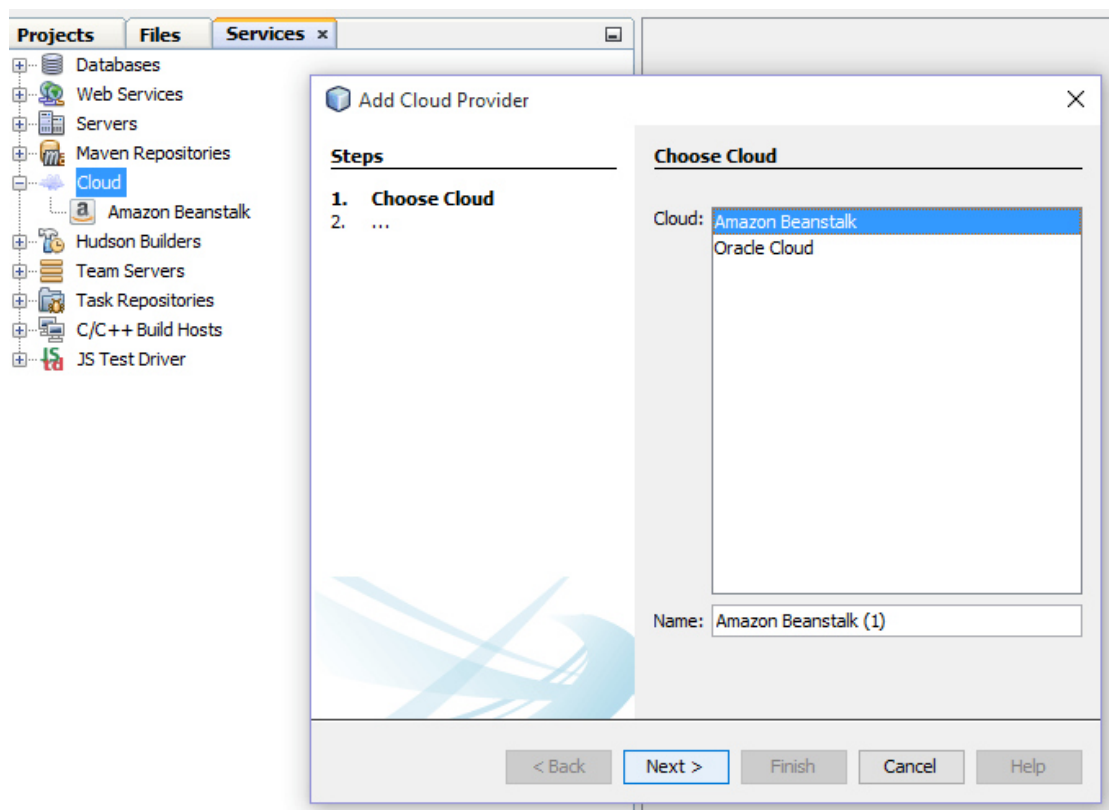


Рисунок 3.1 – Подключение среды разработки к AWS

Создать новое приложение в Builder, выбрав тип приложения и нажав кнопку «Создать новое приложение», выберите Bootstrap JS App, введите имя приложения и нажмите «Создать» (рисунок 3.2).

Create new app



From template

From example

From backup

☐ JQuery App

☒ Bootstrap JS App

Use Bootstrap to build responsive web apps only. Requires basic knowledge of JS. Read more [here](#) if you need help deciding which template to use.

☐ EXAMPLE

☐ Contacts

Рисунок 3.2 – Выбор типа приложения

Откроется конструктор приложений. В приложении уже есть страницы index и Screen1 по умолчанию. Для примера работы требуется создать еще две страницы и сделать подключение к облаку и базе данных. В конструкторе приложений перейдите на вкладку Create new>Page. Добавить Details и PizzaCreator страницы. Переименовать Screen1 в Menu.

Так как корзина и количество товара являются переменными значениями, требуется создать переменные для их хранения, для этого требуется перейти в меню App Settings > Storage and Model > Model. Добавьте следующие элементы модели: Array - тип Array, Object –тип Object: Предварительный просмотр изменений (рисунки 3.3 и 3.4).

Model

Storage

Name	Type
<div> <div>▼</div> <div>Array</div> <div>Array[1]</div> </div>	<div> <input type="button" value="Add"/> <input type="button" value="Clone"/> <div>Array ▼</div> <input type="button" value="Delete"/> </div>
<div> <div>Object</div> </div>	<div> <input type="button" value="Add"/> <div>String ▼</div> <input type="button" value="Delete"/> </div>
<div> <div>Object</div> </div>	<div> <input type="button" value="Add"/> <div>Object ▼</div> <input type="button" value="Delete"/> </div>

Рисунок 3.3 – Формирование переменных

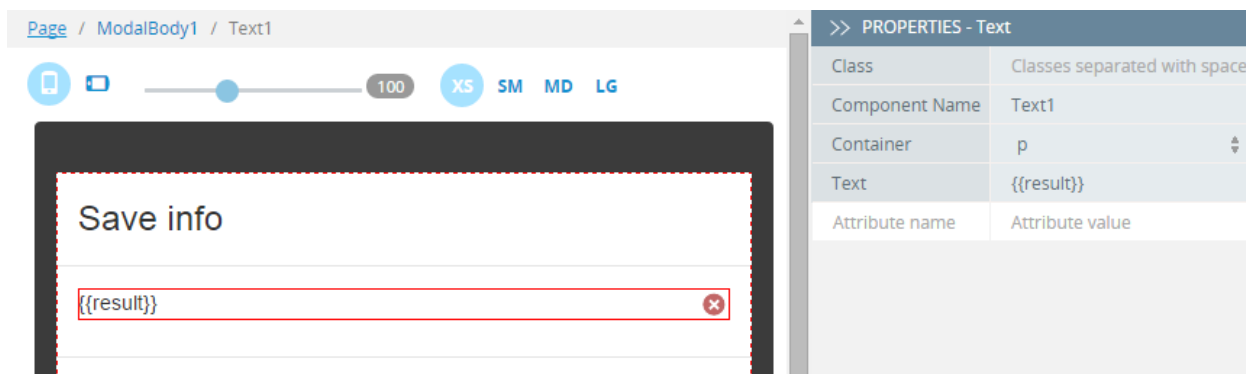


Рисунок 3.4 – Связывание переменной с элементами интерфейса

Маршрутизация необходима для настройки отображения содержимого страниц и скриптов AWS.

Перейти к Project>Routing. Введите pizzaCreator в поле Name и выберите PizzaCreator как Page. Нажмите кнопку Добавить. Создайте следующую маршрутизацию таким же образом (рисунок 3.5).

Name:	<input type="text"/>	Page:	<input type="text" value="PizzaCreator"/>	<input type="button" value="Add"/>
Route name	Page	Default route	Path	
<input type="text" value="details"/>	Details	☐	<input type="text" value="/Details.html/:id"/>	<input type="button" value="Delete"/>
<input type="text" value="menu"/>	Menu	☒	<input type="text" value="/Menu.html"/>	<input type="button" value="Delete"/>
<input type="text" value="pizzaCreator"/>	PizzaCreator	☐	<input type="text" value="/PizzaCreator.html"/>	<input type="button" value="Delete"/>

Рисунок 3.5 – Связывание страниц интерфейса со скриптами

Также требуется протестировать связь среды IDE NetBeans с базой данных, для чего воспользуемся меню «Сервисы». При нажатии на кнопку «Тест», появится все доступные скрипты (рисунок 3.6) и записи базы данных (рисунок 3.7)

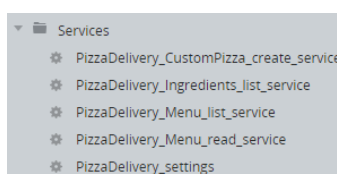


Рисунок 3.6 – Доступные скрипты

☐	_id	name	price	about	consist	ima
		string	number	string	array	file
☐	54d25b6e975...	Tropical chicken	7.7	Sliced chicken r...	["Sliced chicken","Pineapples","Tomato sau...	
☐	54d25715975...	Island Supreme	12.3	Tuna, onions, p...	["Tuna","Onions","Pineapples","Crabmeat ...	
☐	54d2561f975...	BBQ Chicken	8.9	Shredded chick...	["Shredded chicken","Mozzarella","Mushro...	
☐	54bff8cd975a...	Meat Lover's Pesto	14.6	Crisp on the ou...	["Ground beef","Beef pepperoni","Sliced c...	
☐	54bff7fe975a...	Margarita	7	Double mozzar...	["Mozzarella","Mozzarella","Tomatoes","To...	
☐	54bff864975a...	Veggie Lover	8.6	Mushrooms, o...	["Mushrooms","Onions","Capsicums","Tom...	
☐	54c39f41975a...	Pesto Chicken	11.9	Baked fresh da...	["Chicken pepperoni","Shredded chicken","...	
☐	54c39f44975a...	Deluxe Cheese	5.2	Double mozzar	["Mozzarella","Mozzarella","Tomato sauce"]	

Рисунок 3.7 – База данных DinamoDB

После настройки связи со скриптами на сервере и СУБД, перейдем к верстке страницы.

Для чего откроем шаблон страницы с уже имеющимися элементами по умолчанию (рисунок 3.8).

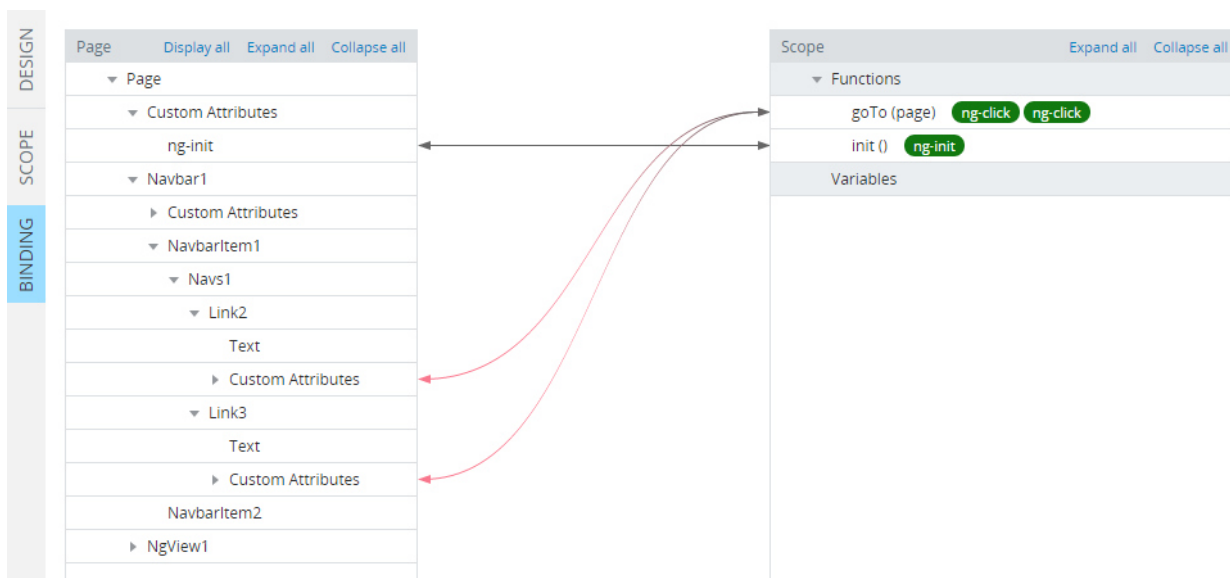


Рисунок 3.8 – Формат страницы по умолчанию

Далее по сетке формируем дизайн элементов приложения (рисунки 3.9 и 3.10).

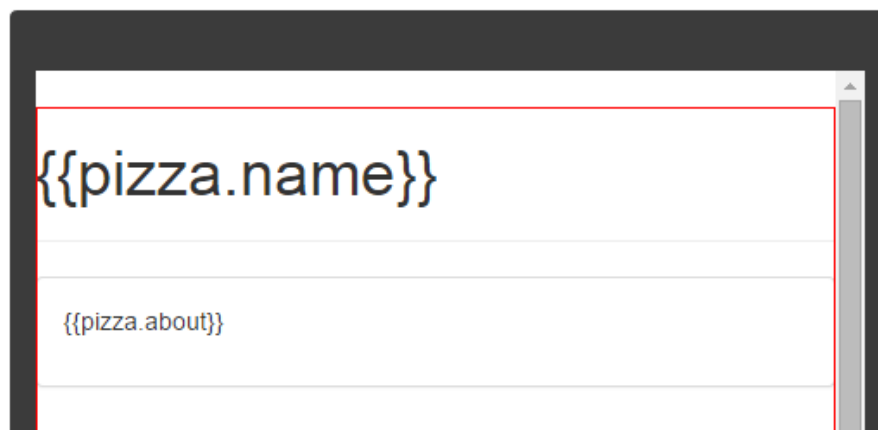


Рисунок 3.9 – Работа с элементами меню на странице

Name	Price	Vol
{{item.name}}	{{item.price currency}}	{{item.vol}} gm
{{customPizza.price currency}}		

Рисунок 3.10 – Формирование рабочих элементов

После формирования элементов, необходимо добавить логику на стороне клиента, для чего используется классические javascript вставки.

Переключитесь на раздел SCOPE вкладки страницы PizzaCreator. Добавьте переменную customPizza типа Object. Добавьте переменную customList типа Array. Используйте следующий код для функции init():

```

$scope.customPizza.price = 0;
$scope.customPizza.consist = [];
Apperyio.get("PizzaDelivery_Ingredients_list_service")().then(function(success) {
    $scope.list = success.data;
    $scope.selected = $scope.list[0];
}, function(error) {
    console.log(error);
});

```

Добавьте функцию addItem()со следующим кодом:

```

$scope.customList.push($scope.selected);
$scope.customPizza.price += $scope.selected.price;
$scope.customPizza.consist.push($scope.selected.name);

```

Добавьте функцию savePizza()со следующим кодом:

```

$scope.customPizza.about = "Custom pizza";
$scope.customPizza.name = "Custom pizza name";

Apperyio.get("Pizza
service")({data:$scope.customPizza}).then(function(success) {
    Apperyio.get('Modals').loadModal("Modal").then(
        function(modal) {
            modal.scope.current_item = {
            };

            modal.scope.result = true;
            modal.scope.modalInstance = modal.open({
                size: '',
                keyboard: true,
                backdropClass: '',
                windowClass: '',
                windowTemplateUrl: ''
            });

            modal.scope.modalInstance.result.then(
                function(current_item) {
                },
                function(error) {
                    console.log(error);
                }
            );
        }
    );
}

```

```

        );
    });
},
function(error) {
    console.log(error);
    Apperyio.get('Modals').loadModal("Modal").then(
        function(modal) {
            modal.scope.current_item = {
            };

            modal.scope.result = "error";
            modal.scope.modalInstance = modal.open({
                keyboard: true,          /* true | false */
                windowClass: '',         /
                windowTemplateUrl: '' /
            });

            modal.scope.modalInstance.result.then(
                function(current_item) {
                },
                function(error) {
                    console.log(error);
                }
            );
        });
    });
});

```

Проверить привязку скриптов на стороне клиента можно просмотром новых связей на странице, как показано на рисунке 3.11.

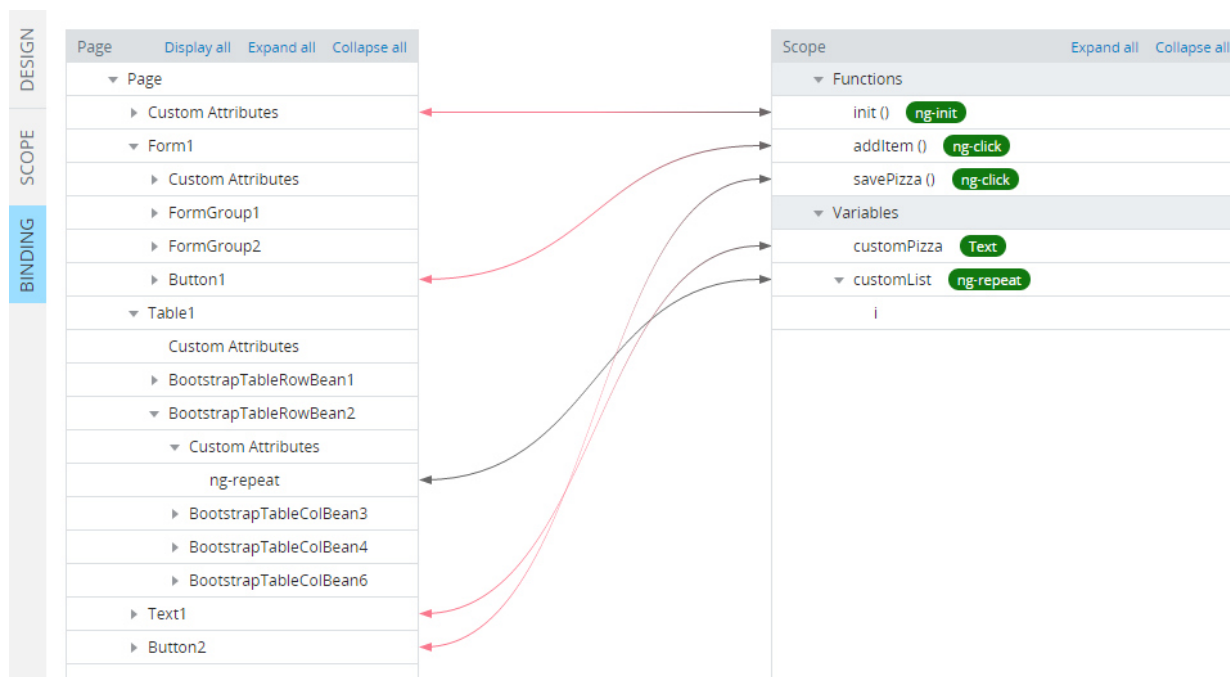


Рисунок 3.11 – Формат страницы после добавления элементов и скриптов

В результате запуска страницы на проверку в IDE NetBeans получим следующую картинку (рисунок 3.12).

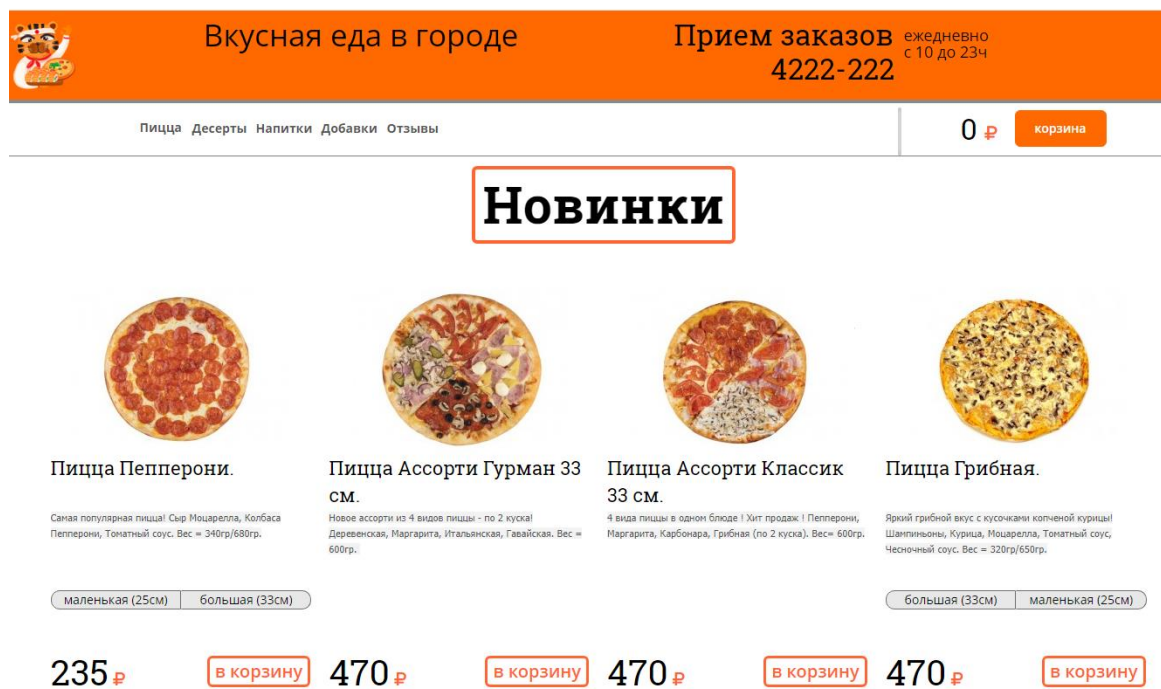


Рисунок 3.12 – Результат связывания элементов интерфейса

Как видим появились все данные страницы при запуске приложения на локальной машине. Это говорит о том, что связывание web интерфейса и бессерверной архитектуры произошло успешно.

Оставшиеся части, в том числе и динамические, реализуются в IDE NetBeans по аналогии со способами, описанными выше.

					ПГУ 1.090501.17.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

4 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ ИНВЕСТИЦИЙ ВО ВНЕДРЕНИЕ WEB СЕРВИСА ЭЛЕКТРОННОЙ КОММЕРЦИИ НА БАЗЕ БЕССЕРВЕРНОЙ АРХИТЕКТУРЫ

4.1 Обоснование выбора объекта

Расчет затрат на разработку WEB сервис будет производиться в соответствии с методическим указанием. В процессе дипломного проектирования было проведено внедрение WEB сервиса электронной коммерции на базе бессерверной архитектуры. Эта система позволит повысить эффективность труда, снизить количество ошибок при ведении учета, а также выполнять хранение и обработку данных с высокой скоростью и надежностью.

Капитальные вложения во внедрение системы следует рассматривать, как инвестиции, необходимые для получения прибыли. Поэтому возникает необходимость в проведении оценки эффективности инвестиционного проекта разработки.

Среди модернизаций системы можно выделить создание отчетных форм, объектов базы данных и пользовательских процедур. Кроме того, необходимо закупить лицензию на требуемое количество рабочих мест.

Над модернизацией модулей сервиса «Amazon» для WEB сервиса электронной коммерции работали следующие специалисты:

- ведущий специалист, знающий предметную область и формирующий задания на модернизацию;
- системный аналитик – технический консультант по технической части разработки ПО;

- программист, непосредственно занимающийся разработкой и доработкой ПО, а также выполняющий функции менеджера проекта;
- руководитель проекта, выполняющий функции по администрированию и ведению нормативно-справочной информации и общему курированию проекта.

Разработка будет вестись в помещениях и на оборудовании заказчика.

В таблице 4.1 приведены данные для расчета затрат на модернизацию и внедрение WEB сервиса.

Таблица 4.1 – Данные для расчета затрат на внедрение WEB сервиса

Показатели, единица измерения	Значение показателя	Источник информации
Страховые взносы, %	30	В соответствии с НК РФ
Социальное страхование, %	2,9	В соответствии с НК РФ
Медицинское страхование, %	5,1	В соответствии с НК РФ
В пенсионный фонд, %	22	В соответствии с НК РФ
Эксплуатационные расходы до внедрения системы, руб.	320 000	По данным команды разработчиков
Накладные расходы, %	20	По данным команды разработчиков
Стоимость 1кВт/ч, руб.	3,44	По данным команды разработчиков
Стоимость одного часа работы системы, руб.	15	По данным команды разработчиков
Время разработки программы, месяц	2	По согласованию заказчика и исполнителя

Продолжение таблицы 4.1

Показатели, единица измерения		Значение показателя	Источник информации
Заработная плата работающих, руб.		Время работы, раб. дни	По данным команды разработчиков
Ведущий специалист	20000	20	
Системный аналитик	16000	10	
Программист	30000	35	
Руководитель проекта	40000	15	

4.2 Расчет затрат на модернизацию WEB сервиса и покупку лицензий

Затраты на модернизацию программы складываются из затрат на оплату труда специалистов с отчислениями на социальные нужды, в которые входят основная заработная плата разработчиков, страховые взносы, накладные расходы организации, где разрабатывается программа. Также необходимо учесть затраты на закупку лицензий на модули системы.

4.2.1 Расчет затрат на оплату труда

В таблице 4.2 приведены данные для расчета фонда оплаты труда в соответствии с таблицей 4.1.

Таблица 4.2 – Данные для расчета фонда оплаты труда

Специалист	Месячная заработная плата, руб.	Заработная плата в день, руб.
Ведущий специалист	20 000	909
Системный аналитик	16 000	727
Программист	30 000	1 363
Руководитель проекта	40 000	1 818

Заработная плата в день рассчитывается путем деления месячной заработной платы специалиста на 22 рабочих дня. В таблице 4.3 приведен расчет фонда оплаты труда.

Таблица 4.3 – Расчет фонда оплаты труда

Специалист	Время работы, раб. дни	Сумма оплаты, руб.
Ведущий специалист	20	18 180
Системный аналитик	10	7 270
Программист	35	47 705
Руководитель проекта	15	27 270
Всего		100 425

Сумма оплаты рассчитывается путем умножения заработной платы в день на отработанное время в соответствии с таблицей 4.1.

4.2.2 Расчет отчислений на социальные нужды

В таблице 4.4 приведен расчет страховых взносов, а именно на социальное страхование, обязательное медицинское и пенсионное страхование. Страховые взносы рассчитываются путем умножения заработной платы на соответствующую ставку.

Таблица 4.4 – Расчет страховых взносов

Специалист	Страховые взносы (30%), руб.	Социальное страхование (2,9%), руб.	Медицинское страхование (5,1%), руб.	Пенсионное страхование (22%), руб.
Ведущий специалист	6 181.20	527.22	927.18	4 726.80
Системный аналитик	2 471.80	210.83	370.77	1 890.20
Программист	16 219.70	1 383.45	2 432.96	12 403.30
Руководитель проекта	9 271.80	790.83	1 390.77	7 090.20
Всего	34 144.50	2 912.33	5 121.68	26 110.50

4.2.3 Расчет накладных расходов

По данным команды разработчиков накладные расходы составляют 20% от затрат на оплату труда.

$$\text{Накладные расходы организации } P_{\text{НО}} = \frac{\alpha_{\text{НО}} * Z_0}{100},$$

где $\alpha_{\text{НО}}$ – процент накладных расходов организации;

Z_0 – основная заработная плата.

Таким образом, расходы на оплату труда составляют 100 425 рублей, накладные расходы равны 20 085 рублей.

4.2.4 Расчет амортизационных отчислений

Для работы специалистов были использованы 4 компьютера стоимостью 20 000 рублей каждый. Компьютеры принадлежат заказчику. Расчет амортизационных отчислений зависит от выбранного способа расчета амортизации и от срока полезного использования. В этой работе расчет будет производиться линейным способом:

$$A_{\Gamma} = \frac{OC_{\Pi} * H_a}{100},$$

где OC_{Π} – первоначальная стоимость компьютера,

H_a – годовая норма амортизационных отчислений.

$$H_a = \frac{100}{n},$$

где n – количество лет срока полезного использования.

Срок полезного использования компьютера составляет 5 лет.

Следовательно, годовая норма амортизационных отчислений равна:

$$H_a = \frac{100}{5} = 20\%.$$

Таким образом, амортизация компьютера составит:

$$A_{\Gamma} = \frac{20000 * 20\%}{100\%} = 4\,000 \text{ рублей.}$$

Так как работы велись по пятидневной рабочей неделе, то в году получается 258 рабочих дней. Дневная сумма амортизации рассчитывается путем деления суммы амортизационных отчислений за год на сумму рабочих дней в году:

$$\frac{4000}{258} = 15,5 \text{ рублей.}$$

Ежемесячная сумма амортизационных отчислений равна:

$$\frac{4000}{12} = 333,33 \text{ рубля.}$$

В таблице 4.5 приведены расходы на амортизацию.

Таблица 4.5 – Расходы на амортизацию

Компьютер	Время работы, раб. дни	Амортизационные отчисления, руб.
Компьютер ведущего специалиста	20	310
Компьютер системного аналитика	10	155
Компьютер программиста	35	542,5
Компьютер руководителя проекта	15	232,5
Всего		1 240

Таким образом, сумма затрат на амортизацию составит 1 240 рублей.

4.2.5 Расчет затрат на обслуживание оборудования

В соответствие с таблицей 4.1 затраты на обслуживание оборудования составляют 15% от амортизационных отчислений и равны:

$$1\,240 * 0,15 = 186 \text{ рублей.}$$

4.2.6 Расчет затрат на электроэнергию

В соответствие с паспортными данными компьютеров, их энергопотребление составляет 0,5 кВт/час. Стоимость одного кВт/час энергии составляет 3,5 рубля, компьютеры работают 8 часов в день. Тогда стоимость электроэнергии, потребляемой одним компьютером, составит:

$0,5 \cdot 3,5 \cdot 8 = 14$ рублей в день. Затраты электроэнергии для всех компьютеров сведены в таблицу 4.6.

Таблица 4.6 – Расчет затрат на электроэнергию

Компьютер	Время работы, раб. дни	Расходы на электроэнергию, руб.
Компьютер ведущего специалиста	20	280
Компьютер системного аналитика	10	140
Компьютер программиста	35	490
Компьютер руководителя проекта	15	210
Всего		1 120

Общие затраты на электроэнергию за время внедрения проекта составят 1 120 рублей.

4.2.7 Расчет стоимости модулей сервиса «Amazon»

В таблице 4.7 приведен расчет стоимости приобретения лицензий сервиса «Amazon» в соответствии с ценовой политикой корпорации «Amazon».

Таблица 4.7 – Расчет затрат на приобретение лицензий сервиса «Amazon»

Модуль	Количество рабочих мест, шт.	Цена, руб.	Сумма, руб.
«Администратор»	1	10 000	10 000
«Бухгалтерский учет» серверная часть	1	10 000	10 000
«Бухгалтерский учет» клиентская часть	14	7 000	98 000
«Закупки, склад, реализация» серверная часть	1	10 000	10 000
«Закупки, склад, реализация» клиентская часть	11	70 000	77 000
«Управление финансами» серверная часть	1	10 000	10 000
«Управление финансами» клиентская часть	12	7 000	84 000
Всего			299 000

В таблице 4.8 приведена общая сумма затрат на модернизацию сервиса «Amazon».

Таблица 4.8 – Расчет затрат на модернизацию сервиса «Amazon»

Статья затрат	Сумма, руб.
Заработная плата	100 425
Страховые взносы	34 144,50
в том числе:	
Социальное страхование	2 912
Обязательное медицинское	5 121,68
В пенсионный фонд	26 110,50
Накладные расходы	20 085
Обслуживание оборудования	26 112
Амортизационные отчисления	1 240
Электроэнергия	1 120
Покупка лицензий	299 000
Всего	482 126,50

4.2.8 Расчет цены программы

Норматив рентабельности, учитывающий прибыль организации, внедряющей систему автоматизации, равен 15%. Отсюда цена программы будет равна:

$$Ц_{\Pi} = 482\,126,50 * 1,15 = 554\,445.48 \text{ руб.}$$

4.3 Расчет эксплуатационных расходов

Расходы, связанные с эксплуатацией программы (P_{Σ}) в месяц, определяются по формуле:

$$P_{\Sigma} = T_{\text{МВ}} * P_{\text{rm}} + A_{\text{ПР}},$$

где T_{MB} – продолжительность машинного времени ЭВМ, используемой в течение месяца для решения задач с помощью сервиса «Amazon», машино-час/месяц потребителя программы;

P_{rm} – эксплуатационные расходы, приходящиеся на 1 час машинного времени этой ЭВМ, руб/машино-час;

$A_{пр}$ – амортизационные отчисления с этой программы (руб/час).

Амортизационные отчисления находятся по формуле:

$$A_{пр} = Ц_{п} * N_A / 100$$

Срок полезного использования программы равен 5 лет. Таким образом, амортизация в год составит:

$$A_{г} = \frac{554\,445.48 * 20\%}{100\%} = 110\,889.10 \text{ руб.}$$

Сумма ежемесячных амортизационных отчислений составит:

$$A_{м} = \frac{A_{г}}{12} = \frac{110\,889.10}{12} = 9\,240.76 \text{ руб.}$$

АСУ будет эксплуатироваться 8 часов в сутки 258 дней в году. Тогда продолжительность машинного времени в год составит:

$$8 \text{ ч} * 258 \text{ дн.} = 2064 \text{ машино-час/год.}$$

Продолжительность машинного времени в месяц составит:

$$\frac{2064}{12} = 172 \text{ машино-час/месяц.}$$

Эксплуатационные расходы, приходящиеся на 1 час машинного времени ЭВМ в соответствие с таблицей 4.1 составят 15 руб/машино-час. В итоге, эксплуатационные расходы в месяц после внедрения системы составят:

$$172 * 15 + 9\,240.76 = 11\,820.76 \text{ руб.}$$

4.4 Расчет денежного потока

В целях экономического обоснования эффективности вложений необходимо произвести расчет показателей:

- чистого дисконтированного дохода (ЧДД);
- срока окупаемости вложений (Ток);
- индекса рентабельности (IP).

Денежный поток за первый месяц определяется как сумма затрат за первые 22 дня, в соответствие с данными команды разработчиков эта сумма составляет 390 563.25 рублей. Денежный поток за 2 месяц определяется как сумма затрат за оставшиеся дни. По данным команды разработчиков эта сумма составит 91 563.25 рублей.

Амортизационные отчисления представлены в виде суммы отчислений за оборудование и ПО:

$$A = A_{\text{ЭВМ}} + A_{\text{ПО}} = 333,33 + 11\,820.76 = 12\,154.09 \text{ руб./месяц.}$$

Коэффициент дисконтирования K определяется так:

$$K = \frac{1}{(1+E)^t},$$

где t – номер месяца,

E – коэффициент (ставка) дисконтирования, в данной работе она принимает значение 11%.

$$K_1 = \frac{1}{(1 + 0.11)^1} = 0.901$$

$$K_2 = \frac{1}{(1 + 0.11)^2} = 0.812$$

$$K_3 = \frac{1}{(1 + 0.11)^3} = 0.731$$

$$K_4 = \frac{1}{(1 + 0.11)^4} = 0.659$$

$$K_5 = \frac{1}{(1 + 0.11)^5} = 0.594$$

$$K_6 = \frac{1}{(1 + 0.11)^6} = 0.535$$

$$K_7 = \frac{1}{(1 + 0.11)^7} = 0.482$$

$$K_8 = \frac{1}{(1 + 0.11)^8} = 0.434$$

$$K_9 = \frac{1}{(1 + 0.11)^9} = 0.391$$

$$K_{10} = \frac{1}{(1 + 0.11)^{10}} = 0.352$$

$$K_{11} = \frac{1}{(1 + 0.11)^{11}} = 0.317$$

$$K_{12} = \frac{1}{(1 + 0.11)^{12}} = 0.286$$

...

$$K_{24} = \frac{1}{(1 + 0.11)^{24}} = 0.082$$

Коэффициент K всегда меньше единицы, так как сумма денег, находящаяся на руках сегодня, больше той же суммы в будущем. ЧДД рассчитывается путем произведения денежного потока на коэффициент дисконтирования. Данные расчета приведены в таблице 4.9.

Таблица 4.9 – Расчет ЧДД

№ ме- ся- ца	Инвест и-ции, руб.	Текущие эксплуатационные расходы, руб.			Амор- тиза- ция, руб.	Денеж- ный поток, руб.	Ко- эффи- циент дис- кон- тиро- вания	ЧДД, руб.
		До внед- рения	Пос-ле внед- рения	Эконо- мия				
1	390 563.25	0	0	0	0	-390563.25	0.901	-351 858.78
2	91 563.25	0	0	0	0	-91563.25	0.812	-74 314.79
3	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.731	102 610.52
4	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.659	92 441.91
5	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.593	83 281.00
6	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.535	75 027.93
7	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.482	67 592.73
8	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.434	60 894.35
9	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.391	54 859.77
10	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.352	49 423.22
11	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.317	44 525.42
12	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.286	40 112.99
13	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.258	36 137.83
14	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.232	32 556.61
15	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.209	29 330.28
16	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.188	26 423.67
17	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.170	23 805.11
18	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.153	21 446.04
19	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.138	19 320.76
20	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.124	17 406.09
21	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.112	15 681.16
22	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.101	14 127.17
23	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.091	12 727.18
24	0	140 000	11 820.76	128 179.24	12 154.09	140 333.33	0.082	11 465.93
								505 024.13

Исходя из расчета ЧДД видно, что денежный поток на первых 2-х месяцах разработки отрицательный, а начиная с 3-го месяца происходит экономия средств от использования системы. ЧДД положительный, поэтому инвестиционный проект жизнеспособен. Индекс рентабельности (доходности) определяется отношением суммы приведенных эффектов к размеру капитальных вложений:

$$IP = \frac{505\,024,13}{482\,126,50} = 1.05$$

Если $IP > 1$, то проект эффективен, иначе – нет. Индекс рентабельности проекта равен 1,05, следовательно, можно сделать вывод, что данный инвестиционный проект эффективен.

Срок окупаемости – это период времени, который потребуется для возмещения инвестиций. Он равен отношению затрат на разработку к сумме месячной экономии.

$$T_{OK} = \frac{482\,126,50}{128\,179,24} = 3.76 \text{ мес.}$$

Срок окупаемости составляет 3.76 месяца. Дальнейшее использование системы несет заказчику чистую прибыль.

Исходя из полученных результатов, можно сделать вывод, что внедренная и модернизированная система жизнеспособна и эффективна при использовании, ее внедрение и эксплуатация дает реальную экономию за счет сокращения времени на сбор и обработку информации, система окупается за 3.76 месяца, а дальнейшая ее эксплуатация приносит заказчику чистую прибыль. Общая стоимость проекта будет составлять 505 024,13 рубля с учетом прибыли организации-исполнителя. Отчисления страховых взносов составляют 34 144,50 рублей. Внедрение программы будет иметь экономический эффект из-за того, что индекс рентабельности больше единицы.

5 ПСИХОФИЗИОЛОГИЧЕСКИЕ ФАКТОРЫ, ВОЗДЕЙСТВУЮЩИЕ НА ОПЕРАТОРА

Психология безопасности труда оператора ПЭВМ составляет важное звено в структуре мероприятий по обеспечению безопасной деятельности человека. Психофизиологические факторы можно разделить на две группы: физические и нервно-психические перегрузки.

5.1 Физические факторы

Физическая нагрузка может быть связана с перемещением материалов, ЭВМ, техники и т.п. на необходимые расстояния и обуславливать динамическую перегрузку.

Статическая нагрузка обусловлена необходимостью работающему оператору прилагать усилия без перемещения всего тела или отдельных частей тела. Она определяется весом удерживаемого груза (величиной прилагаемого усилия) и временем удержания.

При выполнении трудовых функций оператор находится в вынужденной позе (сидя за столом, держа руки и спину в определенной напряженной позе). При проектировании рабочего места следует учитывать, что фиксированная рабочая поза физиологически не оправдана, так как она вызывает нарушение кровообращения в нижних конечностях и органах тазовой области, приводящие к профессиональным заболеваниям (синдром запястного канала, заболевания суставов и др.).

При проектировании рабочих мест необходимо стремиться к тому, чтобы рабочая поза была как можно ближе к естественной позе человека. Поэтому целесообразно предусматривать возможность работы как стоя, так и сидя.

Особого внимания заслуживает проектирование кресел для лиц, постоянно выполняющих работу с ЭВМ. Конструкция кресла должна быть такой, чтобы как можно равномернее распределить давление тела на площадь опоры. Это возможно тогда, когда кресло в наибольшей степени соответствует анатомическому строению человека.

Отрицательно на состояние здоровья работающих отражается гиподинамия - нарушение функций организма (опорно-двигательного аппарата, кровообращения, дыхания, пищеварения и др.) при ограниченной двигательной активности, снижении сил сопротивления мышц.

Профилактика гиподинамии предусматривает исключение статической работы, изменение рабочей позы в процессе работы, проведение производственной гимнастики с рациональным комплексом физических упражнений и т.п. Организация комнат психологической разгрузки способствует снижению усталости и повышению производительности труда работающих, улучшает их настроение, что в конечном счете способствует сохранению их работоспособности и обеспечению охраны труда.

Напряженность внимания характеризуется длительностью сосредоточения наблюдения, числом объектов одновременного наблюдения, плотностью сигналов (световых, звуковых) и сообщений. Такой характер деятельности присущ работающим с видео дисплейными терминалами, водителям транспортных средств, поездным диспетчерам, операторам, наблюдающим за сигнализацией на пульте управления, и др.

Степень напряженности анализаторских функций для зрительного анализатора зависит от размера объекта различения и расстояния объекта от глаз, различия в контрастности объекта различения и фона, для слухового анализатора – от соотношения между уровнями речи и шума.

5.2 Тяжесть и напряженность труда

Тяжесть труда является количественной характеристикой физического труда. Напряженность труда – количественная характеристика умственного труда. Она определяется величиной информационной нагрузки.

На производстве различают четыре уровня взаимодействия факторов условий труда на человека:

- комфортные условия труда обеспечивают оптимальную динамику работоспособности человека и сохранение его здоровья;
- относительно дискомфортные условия труда при воздействии в течение определённого интервала времени обеспечивают заданную работоспособность и сохранение здоровья, но вызывают субъективные ощущения и функциональные изменения, не выходящие за пределы нормы;
- экстремальные условия труда приводят к снижению работоспособности человека, не вызывают функциональные изменения, выводящие за пределы нормы, но не ведущие к патологическим изменениям;
- сверх экстремального условия труда приводят к возникновению в организме человека патологических изменений и к потере трудоспособности.

Медико-физиологическая классификация тяжести и напряженности труда проводится на основании комплексной количественной оценки факторов условий труда, называемой интегральной величиной тяжести и напряженности труда.

5.3 Психологические факторы

В психической деятельности оператора ПЭВМ различают три основные группы компонентов: психические процессы, свойства и состояния.

Психические процессы составляют основу психической деятельности и являются динамическим отражением действительности. Без них невозможно формирование знаний и приобретение жизненного опыта.

Психические свойства – это свойства личности или её существенные особенности (направленность, характер, темперамент). Среди качеств личности выделяют интеллектуальные, эмоциональные, волевые, моральные, трудовые. Эти свойства устойчивы и постоянны.

Психические состояния человека – это относительно устойчивая структурная организация всех компонентов психики, выполняющая функцию активного взаимодействия человека с внешней средой, представленной в дан-ней момент конкретной ситуацией. Психические состояния отличаются разнообразием и временным характером, определяют особенности психической деятельности в конкретный момент и могут положительно или отрицательно сказываться на течении всех психических процессов. Исходя из задачи психологии труда и проблем психологии безопасности труда целесообразно выделять производственные психические состояния и особые психические состояния, имеющие важное значение в организации профилактики производственного травматизма и предупреждения аварийности.

5.4 Психические состояния

Психические состояния возникают в процессе трудовой деятельности оператора ПЭВМ, классифицируются по следующим группам:

По признаку преобладания одной из сторон психики различают состояния эмоциональные, волевые; состояния, в которых доминируют процессы восприятия и ощущения; состояния внимания (рассеянность, сосредоточенность); состояния, для которых характерно мыслительная активность и т.д.

Наиболее важным является рассмотрение состояний по уровню напряжения, так как именно этот признак наиболее существенен с точки зрения влияния на эффективность и безопасность деятельности.

Умеренное напряжение – нормальное рабочее состояние, возникающее под мобилизующим влиянием трудовой деятельности. Это состояние психической активности – необходимое условие успешного выполнения действий. Оно сопровождается умеренным изменением физиологических реакций организма, проявляется в хорошем самочувствии, стабильном и уверенном выполнении действий. Умеренное напряжение соответствует работе в оптимальном режиме. Оптимальный режим работы осуществляется в комфортных условиях, при нормальной работе технических устройств. Обстановка является привычной, рабочие действия осуществляются в строго определённом порядке, мышление носит алгоритмический характер. В оптимальных условиях промежуточные и конечные цели труда достигаются при невысоких нервно-психических затратах. Обычно здесь имеют место длительное сохранение работоспособности, отсутствие грубых нарушений, ошибочных действий, отказов, срывов и других аномалий. Деятельность в оптимальном режиме характеризуется высокой надёжностью и оптимальной эффективностью.

Повышенное напряжение сопровождает деятельность, протекающую в экстремальных условиях. Экстремальные условия – условия, требующие от работающего максимального напряжения физиологических и психических функций, резко выходящего за пределы физиологической нормы. Экстремальный режим – это режим работы в условиях, выходящих за пределы оптимума. Отклонения от оптимальных условий деятельности требуют повышенного волевого усилия или, иначе говоря, вызывают напряжение. Неблагоприятные факторы, повышающие напряжение, следующие:

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		90

- 1) физиологический дискомфорт, т.е. несоответствие условий обитания нормативным требованиям;
- 2) биологический страх;
- 3) дефицит времени на обслуживание;
- 4) повышенная трудность задачи;
- 5) повышенная значимость ошибочных действий;
- 6) наличие релевантных помех;
- 7) неуспех вследствие объективных обстоятельств;
- 8) дефицит информации для принятия решений;
- 9) недогрузка информацией (сенсорная деривация);
- 10) перегрузка информацией;
- 11) конфликтные условия, т.е. условия, при которых выполнение одного из них требует осуществления действий, противоречащих выполнению другого условия.

Напряжения могут быть классифицированы в соответствии с теми психическими функциями, которые преимущественно вовлечены в профессиональную деятельность и изменения которых наиболее выражены в неблагоприятных условиях.

Интеллектуальное напряжение – напряжение, вызванное частым обращением к интеллектуальным процессам при формировании плана обслуживания, обусловленное высокой плотностью потока проблемных ситуаций обслуживания.

Монотония – напряжение, вызванное однообразием выполняемых действий, невозможностью переключения внимания, повышенными требованиями как к концентрации, так и к устойчивости внимания.

Эмоциональное напряжение – напряжение, вызванное конфликтными условиями, повышенной вероятностью возникновения аварийной ситуации, неожиданностью либо длительным напряжением прочих видов.

Утомление – напряжение, связанное с временным снижением работоспособности, вызванное длительной работой.

Характеристики напряжений, наиболее присущие профессиональной деятельности человека-оператора, следующие:

Состояние утомления. Утомление является одним из самых распространённых факторов, оказывающих существенное влияние на эффективность и безопасность деятельности. Утомление представляет собой весьма сложный и разнородный комплекс явлений. Полное содержание его определяется не только физиологическим, но также психологическим, результативно производственным и социальным факторами. Исходя из этого, утомление и должно рассматриваться по меньшей мере с трёх сторон:

- 1) со стороны субъективной – как психическое состояние;
- 2) со стороны физиологических механизмов;
- 3) со стороны понижения эффективности труда.

Отмеченные психологические показатели утомления проявляются в зависимости от его силы. Бывает слабое утомление, при котором не происходит значительных изменений в психике. Такое утомление только сигнализирует о необходимости принять меры, чтобы не понизилась работоспособность. Вредно переутомление, при котором резко понижается работоспособность и тем самым безопасность деятельности. При переутомлении указанные выше нарушения в психической сфере очень заметны.

В динамике утомления выделяют разные стадии. Первая стадия утомления характеризуется относительно слабым чувством усталости, при этом производительность труда не падает или падает незначительно.

На второй стадии утомления понижение производительности становится заметным и все более и более угрожающим, причем часто это понижение относится только к качеству, а не к количеству выработки.

Третья стадия характеризуется острым переживанием утомления, которая принимает форму переутомления. Кривая работы или резко снижается, или же принимает "лихорадочную" форму, отражающую попытки человека сохранить должный темп работы, который на данной стадии утомления может ускоряться, но оказывается неустойчивым. В конце концов рабочие действия могут быть так дезорганизованы, что человек почувствует невозможность продолжения работы, переживая при этом болезненное состояние.

Состояние монотонности. В процессе деятельности помимо состояния утомления возникает состояние монотонности, отрицательно действующее на работоспособность человека. Психическое состояние переживания монотонности вызывается действительным и кажущимся однообразием выполняемых на работе движений и действий. Под влияние переживания монотонности человек, не умеющий это психическое состояние сдерживать или устранять, становится вялым, безучастным к работе. Состояние монотонности также отрицательно действует на организм работающих, приводя к преждевременному утомлению.

В физиологической основе монотонности лежит тормозящее действие однообразных повторных раздражителей. Понятие монотонности всегда связывается с трудом по выполнения однообразных и кратковременных операций.

Важным вопросом в понимании природы состояния монотонности является разграничение общих и отличительных его черт по сравнению с состоянием утомления. Общее у этих двух состояний то, что оба они отрицательно влияют на работоспособность человека, и оба переживаются как неприятное чувство. Существенное различие заключается в том, что утомление вызывается тяжестью умственной или физической работы, а состояние монотонности может переживаться и при лёгком, совсем

неутомительном труде. Утомление является фазовым процессом, а монотонность характеризуется волновой кривой, обладающей подъемами и спадами. Утомление усиливает психическую напряженность; монотонность наоборот, снижает её.

Необходимо также отличать состояние монотонности от психической возбудимости. Психическая возбудимость вызывает волнение, нервозность, беспокойство; монотонность, напротив, сопровождается полусонным состоянием, сопровождающимся снижением психической активности и скукой. Психическая насыщенность вызывается, главным образом, повторением деятельности, а для появления монотонности необходимы и другие объективные данные (бедность раздражителей, однообразие их, ограниченность поля наблюдения и т.д.). Следует подчеркнуть, что размежевание психического насыщения и монотонности является относительным, так как:

- 1) они взаимно влияют друг на друга;
- 2) их последствия суммировано действуют на состояние человека;
- 3) в производственной практике ни одно из них не встречается в крайних формах.

Эмоциональное напряжение. Оно может по-разному влиять на поведение человека. В соответствии с преобладанием у человека процесса возбуждения или торможения состояние эмоционального напряжения может проявляться в следующих формах поведения в экстремальных условиях.

Напряженный тип поведения проявляется в общей заторможенности, замедленности, скованности, импульсивности и напряженности выполнения рабочих функций. Такие работники кусают губы, лицо перекошено, внимание их приковано к индикатору, на воздействие эмоциогенных факторов реагируют чрезвычайно импульсивно и сильно.

Эмоциональное напряжение может также проявляться в сознательном уклонении человека от выполнения своих функций. В отдельных случаях наблюдается некоторая пассивность и стремление оградить себя от вмешательства в ход событий. Оператор в аварийных условиях испытывает затруднения в организации умственной деятельности, он долго стоит или сидит в застывшей позе, трёт лоб, морщит брови. Здесь находит своё проявление эмоция страха, в результате чего доминирует инстинкт самосохранения; этот тип поведения в экстремальных условиях называется трусливым. Под влиянием страха "трусливые" операторы начинают действовать по привычному, однако не адекватному сложившейся ситуации шаблону.

Тормозной тип эмоционального поведения человека характеризуется полной заторможенностью его действий, вытекающей при воздействии эмоциогенных, необычных и ответственных ситуаций.

Наиболее яркой и опасной формой проявления эмоциональной неустойчивости человека являются аффективные срывы деятельности, в результате чего он начинает действовать агрессивно, бессмысленно и бесконтрольно, что усугубляет состояние управляемой им системы, ускоряя этим наступление катастроф и аварий. Это – агрессивно-бесконтрольный тип поведения.

Существует такая категория людей, которые при наличии надлежащей мотивации, находясь в экстремальных условиях, значительно улучшают показатели своей работы. Такой тип поведения называется прогрессивным. Он характеризуется боевым задором, повышенной работоспособностью, минимальной затратой сил.

Напряженный тип поддаётся исправлению в процессе специально организованного обучения, направленного на формирование навыков. При этом трудовая деятельность на уровне навыка приобретает свойство

стабильности, надёжности и помехоустойчивости. Трусливый тип поведения может быть изменён и улучшен путём определённых воспитательных воздействий, помогающих человеку преодолевать эмоции страха. Так как пока еще не найдены эффективные средства психологического воздействия на представителей тормозного и агрессивно бесконтрольного типов поведения, лучшим путём повышения надёжности систем управления является своевременный отсев таких лиц.

5.5 Влияние режима работы на оператора ЭВМ

Многочисленные исследования влияния на организм человека работы в различные смены показали, что производительность труда, работоспособность, самочувствие, условия травмирования в первую и вторую половину рабочего дня почти одинаковы. В ночное же время труд протекает на фоне снижения функционального состояния центральной нервной системы, на фоне сонливости. Сонливость снижает производительность труда, его качество, а также защитные функции организма человека, т.е. человек становится менее «защищен» от несчастных случаев и аварий. Установлено, что сонливость при работе в ночную смену является, как правило, следствием несоблюдения суточной нормы сна. Люди, работающие в ночные смены, часто мало спят: не более пяти-шести часов, а иногда и того менее. В течение свободного времени, т.е. в дневное время, они занимаются домашней работой, для которой требуется большое напряжение физических сил.

У многих работающих постоянно ночью наблюдается расстройство сна. Люди, работающие в ночные смены, чаще имеют заболевания сердечно-сосудистой и нервной системы. Это объясняется тем, что функции человека в течение суток подвержены закономерным колебаниям. Одни из них активизируются в дневное время и ослабляются ночью, а другие, наоборот,

усиливаются ночью. В организме человека вырабатывается суточная ритмика физиологических функций, характерная значительной инертностью и трудно поддающаяся перестройке.

Работа в ночное время приводит к «рассогласованию» внешнего и внутренних ритмов, что приводит к нервным расстройствам, сопровождающимся, в первую очередь, нарушением сна.

Для профилактики вредного влияния сонливости следует учитывать, что трудовая деятельность оператора связана не только с активностью мышечного аппарата и органов чувств (анализаторов), подчиняющихся сознательному управлению человеком, но и с сердечно-сосудистой, пищеварительной, выделительной, эндокринной и другими системами, которые не находятся под контролем человека и выдерживают суточный ритм активности.

Профилактика сонливости при работе в ночные смены сводится к следующим основным мероприятиям:

- необходимо, чтобы работающие ночью соблюдали суточную норму сна - восемь часов;
- начало смен необходимо устанавливать в 8, 16 и 24 часа при трехсменном режиме работы и в 8, 14, 20, 2 часа при четырехсменном режиме работы;
- переход из смены в смену производить через неделю или две недели, причем чередование смен должно быть таким: утренняя – вечерняя – ночная;
- работающие в ночные смены должны в свободное время не заниматься тяжелым физическим трудом или трудом, требующим напряжения центральной нервной системы. Свободное время должно быть временем активного отдыха.

5.6 Выводы и рекомендации

Работа оператора требует соответствующего напряжения физических и нервных возможностей человека, мобилизации внимания, памяти, требует высокой ответственности за принимаемые в процессе выполнения работы решения. При правильном выборе профессии человек может быстро и эффективно адаптироваться к работе. При отсутствии такого соответствия человек не достигает необходимого мастерства и работает на пределе физиологических возможностей, что и приводит к неврозам, повышенной заболеваемости, аварийности, травматизму и низкой производительности труда. Однако этот человек может легко и быстро овладеть другой профессией, которая больше соответствует психологическим особенностям его организма.

Психофизиологические условия труда определяются также рабочей обстановкой, взаимоотношениями в трудовом коллективе (и особенно между работающими и руководителем), уровнем обслуживания, соответствием выполняемой работы способностям и склонностям работающего, общественной престижностью работы и т.п. Поэтому формы руководства трудовым коллективом играют исключительно важную роль. Организация труда и его безопасность должна строиться на основе учета закономерностей не только физиологических, но и психологических реакций работающего на те или иные условия окружающей его среды. В этих условиях создание благоприятного социального и морального климата в коллективе, психологическая совместимость всех его членов – залог успешного решения не только производственных задач, но и задач охраны труда.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Сухов, Кирилл Node.js. Путеводитель по технологии / Кирилл Сухов. - Москва: ДМК Пресс, 2015. - 416 с.
2. Хэррон, Дэвид Node.js Разработка серверных веб-приложений на JavaScript / Дэвид Хэррон. - Москва: ДМК Пресс, 2016. - 195 с.
3. Хэррон, Дэвид Node.js Разработка серверных веб-приложений на JavaScript / Дэвид Хэррон. - СПб: Питер, 2016. - 144 с.
4. Мартишин С. А., Храпченко М. В. Базы данных. Практическое применение СУБД SQL- и NoSQL-типа для применения проектирования информационных систем. М :Москва, Бином, 2018. - 301 с.
5. Фаулер М., Прамодкумар Дж. Садаладж NoSQL: новая методология разработки нереляционных баз данных, :Москва, Бином, 2018. - 261 с.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ

Файл index.html

```
<!DOCTYPE
PE
html>

<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-
width,initial-scale=1.0">
    <title>pizza</title>
    <link rel="stylesheet"
href="https://cdn.bootcss.com/bootstrap/4.0.0/css/boots
trap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW
/dAiS6JXm" crossorigin="anonymous">
  </head>
  <body>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
  <script
src="https://cdn.bootcss.com/jquery/3.2.1/jquery.slim.m
in.js" integrity="sha384-
KJ3o2DKtIkVYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF9
3hXpG5KkN" crossorigin="anonymous"></script>
  <script
src="https://cdn.bootcss.com/popper.js/1.12.9/umd/popper
r.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvX
usvfa0b4Q" crossorigin="anonymous"></script>
```

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		100

```

<script
src="https://cdn.bootcss.com/bootstrap/4.0.0/js/bootstr
ap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpilMquVdAyjUar5
+76PVCmYl" crossorigin="anonymous"></script>
</html>

```

Файл main.js

```

import NodeRouter from 'node-router'
import Node from 'node'
import App from './App.node'
import { routes } from './routes'
import axios from 'axios'
import {store} from './store/store.js'
axios.defaults.baseURL = "http://localhost:3000/"
Node.prototype.$axios = axios
Node.use(NodeRouter)
const router = new NodeRouter({
  routes,
  mode:'history',
})
new Node({
  el: '#app',
  router,
  store,
  render: h => h(App)
})

```

Файл Component.js

```

sap.ui.define(
[
    "sap/ui/core/UIComponent",
    "sap/ui/Device",

```

```

        "com/app/model/models",
        "com/app/controller/ErrorHandler"
    ], function (UIComponent, Device, models,
ErrorHandler) {
        "use strict";
        return
        UIComponent.extend("com.app.Component", {
            metadata : {
                manifest: "json"
            },
            /**

            * @public
            * @override
            */
            init : function () {

                UIComponent.prototype.init.apply(this,
arguments);

                this._oErrorHandler = new
ErrorHandler(this);

                this.setModel(models.createDeviceModel(),
"device");

                this.setModel(models.createFLPModel(),
"FLP");

                this.getRouter().initialize();

                $.sap.myVar = "global?";

                sap.ui.getCore().AppContext

```

```

= new Object();

        sap.ui.getCore().AppContext.RandomVariable
= "random";

        },
        * @public
        * @override
        */
        destroy : function () {

            this._oErrorHandler.destroy();
                // call the base component's
destroy function

            UIComponent.prototype.destroy.apply(this,
arguments);

        },
        /**
        * @public
        * @return {string} css class,
either

        */
        getContentDensityClass :
function() {

            if

            (this._sContentDensityClass === undefined) {

                if

                (jQuery(document.body).hasClass("sapUiSizeCozy")
||
jQuery(document.body).hasClass("sapUiSizeCompact
")) {

                    this._sContentDensityClass = "";

```

```

        } else if
(!Device.support.touch) { //

        this._sContentDensityClass =
"sapUiSizeCompact";

        } else {

        this._sContentDensityClass =
"sapUiSizeCozy";

        }

    }

    return
this._sContentDensityClass;

    }

    });

}

);

```

Файл AddPizza.controller.js

```

sap.ui.define
([

    "sap/ui/core/mvc/Controller",
    "sap/ui/core/routing/History"
], function(Controller, History) {
    "use strict";
    var globalVar = "globalVar";
    return
Controller.extend("com.app.controller.AddPizza", {
    /**
     * Called when a controller is
    instantiated and its View controls (if available)
    are already created.

```

```

        * Can be used to modify the View before
it is displayed, to bind event handlers and do
other one-time initialization.

        * @memberOf com.app.view.addPizza
        */
onInit: function() {
    console.log($.sap.myVar);

    console.log(sap.ui.getCore().AppContext.Rando
mVariable);
//
    UIComponent.prototype.init.apply(this,
arguments);

    // globalVar = this;

    // var oData = {
    //     recipient: {
    //         name: "World"
    //     }
    // };

    // var oModel = new
JSONModel(oData); //create model
    // this.setModel(oModel,
"userModel"); //set model to component
    },

onNavBack : function() {
    var sPreviousHash =
History.getInstance().getPreviousHash(),
    oCrossAppNavigator =
sap.ushell.Container.getService("CrossApplicationN
avigation");

```

```

        if (sPreviousHash !==
undefined ||
!oCrossAppNavigator.isInitialNavigation()) {
            history.go(-1);
        } else {

            oCrossAppNavigator.toExternal({
                target: {shellHash:
"#Shell-home"}

            });
        }
    },

    onSubmit : function() {
        var oEntry= {};

        var oModel =
this.getView().getModel();

        oEntry.Zcustname=
this.byId("custName").getValue();
        oEntry.Zpname=
this.byId("pizzaType").getSelectedItem().getText()
;
        oEntry.Zppiece=
this.byId("pPiece").getValue();
        oEntry.Zprice = (oEntry.Zppiece *
7.0).toFixed(2);
        oEntry.Zpunit= "EUR";
        // oEntry.Zdate=
sap.ui.model.odata.ODataUtils.formatValue(new
Date(), "Edm.DateTime");

```

```

oModel.read("/OrderListSet",{
    success: function(oData,
responset) {

    oModel.attachRequestCompleted(function() {

        var pizzaID = 0;
        for(var i = 0; i <
oData.results.length; i++) {

            if (Number(oData.results[i].Zoid) > pizzaID )
{

                pizzaID =
Number(oData.results[i].Zoid);

            }

        }
        oEntry.Zoid =
(pizzaID + 1).toString();

        oEntry.Zdate = new
Date();

        oEntry.Ztime =
oEntry.Zdate.getHours() + ":" +
oEntry.Zdate.getMinutes();

        if(oEntry.Zcustname.length > 0 &&
oEntry.Zpname.length > 0 && oEntry.Zppiece > 0 &&
oEntry.Zppiece < 10) {

            oModel.create("/OrderListSet",oEntry);

            var

sPreviousHash =
History.getInstance().getPreviousHash(),

```

```

        oCrossAppNavigator =
sap.ushell.Container.getService("CrossApplicationN
avigation");

                                if
(sPreviousHash !== undefined ||
!oCrossAppNavigator.isInitialNavigation()) {

        history.go(-1);

                                } else {

        oCrossAppNavigator.toExternal({

        target: {shellHash: "#Shell-home"}

                                });

                                }

                                }

                                //

console.log(oEntry);

                                });

                                }

                                });

        }

        /**
         * Similar to onAfterRendering, but this
hook is invoked before the controller's View is
re-rendered
         * (NOT before the first rendering!
onInit() is used for that one!).
         * @memberOf com.app.view.addPizza
         */
        //    onBeforeRendering: function() {
        //
        //    },

```

```

/**
 * Called when the View has been
rendered (so its HTML is part of the document).
Post-rendering manipulations of the HTML could be
done here.

 * This hook is the same one that SAPUI5
controls get after being rendered.
 * @memberOf com.app.view.addPizza
 */
//  onAfterRendering: function() {
//
//  },
/**
 * Called when the Controller is
destroyed. Use this one to free resources and
finalize activities.
 * @memberOf com.app.view.addPizza
 */
//  onExit: function() {
//
//  }
});
});

```

Файл ErrorHandler.js

```

sap.ui.define(
[
    "sap/ui/base/Object",
    "sap/m/MessageBox"
], function (UI5Object, MessageBox) {
    "use strict";
    return

```

```

UI5Object.extend("com.app.controller.ErrorHandle
r", {
    * @class
    * @param
{sap.ui.core.UIComponent} oComponent
    * @public
    * @alias
com.app.controller.ErrorHandler
    */
    constructor : function
(oComponent) {
        this._oResourceBundle =
oComponent.getModel("i18n").getResourceBundle();
        this._oComponent =
oComponent;

        this._oModel =
oComponent.getModel();
        this._bMessageOpen = false;
        this._sErrorText =
this._oResourceBundle.getText("errorText");

        this._oModel.attachMetadataFailed(function
(oEvent) {
            var oParams =
oEvent.getParameters();

            this._showServiceError(oParams.response);
            }, this);

        this._oModel.attachRequestFailed(function
(oEvent) {
            var oParams =
oEvent.getParameters();

```

```

        if
(oParams.response.statusCode !== "404" ||
(oParams.response.statusCode === 404 &&
oParams.response.responseText.indexOf("Cannot
POST") === 0)) {

    this._showServiceError(oParams.response);

    }, this);

},
/**
 * @param {string}
 * @private
 */
_showServiceError : function
(sDetails) {

    if (this._bMessageOpen) {
        return;
    }
    this._bMessageOpen = true;
    MessageBox.error(
        this._sErrorText,
        {
            id :

"serviceErrorMessageBox",

            details: sDetails,
            styleClass:

this._oComponent.getContentDensityClass(),
            actions:

[MessageBox.Action.CLOSE],

            onClose: function

() {

```

```

        this._bMessageOpen = false;
    }.bind(this)
    }
    );
}
});
}
);

```

Файл Object.controller.js

```

sap.ui.de
fine([
    "com/app/controller/BaseController",
    "sap/ui/model/json/JSONModel",
    "sap/ui/core/routing/History",
    "com/app/model/formatter"
], function (
    BaseController,
    JSONModel,
    History,
    formatter
) {
    "use strict";
    return
    BaseController.extend("com.app.controller.Object", {
        formatter: formatter,
        /*
        =====
        ===== */
        /* lifecycle methods
        */
        /*
        =====
        ===== */
    }
    );
}
);

```

```

/**
 * @public
 */
onInit : function () {
    var iOriginalBusyDelay,
        oViewModel = new JSONModel({
            busy : true,
            delay : 0
        });

    this.getRouter().getRoute("object").attachPatternMatch(
        this._onObjectMatched, this);
    iOriginalBusyDelay =
        this.getView().getBusyIndicatorDelay();
    this.setModel(oViewModel,
        "objectView");

    this.getOwnerComponent().getModel().metadataLoaded().then(
        function () {

            oViewModel.setProperty("/delay",
                iOriginalBusyDelay);

        }
    );
},
* @public
*/
onShareInJamPress : function () {
    var oViewModel =
        this.getModel("objectView"),
        oShareDialog =
            sap.ui.getCore().createComponent({
                name:

```

```

"sap.collaboration.components.fiori.sharing.dialog",
        settings: {
            object:{
                id:
location.href,

                share:
oViewModel.getProperty("/shareOnJamTitle")
            }
        }
    });
    oShareDialog.open();
},
* @public
*/
onNavBack : function() {
    var sPreviousHash =
History.getInstance().getPreviousHash(),
    oCrossAppNavigator =
sap.ushell.Container.getService("CrossApplicationNavi
gation");

    if (sPreviousHash !== undefined
|| !oCrossAppNavigator.isInitialNavigation()) {
        history.go(-1);
    } else {

this.getRouter().navTo("worklist", {}, true);
    }
},
* @function
* @param {sap.ui.base.Event}
* @private
*/
_onObjectMatched : function (oEvent) {

```

```

        var sObjectId =
oEvent.getParameter("arguments").objectId;

        this.getModel().metadataLoaded().then(
function() {

            var sObjectPath =
this.getModel().createKey("OrderListSet", {
                Zoid : sObjectId
            });

            this._bindView("/") +
sObjectPath);

            }.bind(this));
    },
    /**
     * @function
     * @param {string}
     * @private
     */
    _bindView : function (sObjectPath) {
        var oViewModel =
this.getModel("objectView"),
        oDataModel =
this.getModel();

        this.getView().bindElement({
            path: sObjectPath,
            events: {
                change:
this._onBindingChange.bind(this),
                dataRequested: function

            }

        }) {

            oDataModel.metadataLoaded().then(function () {

```

```

        oViewModel.setProperty("/busy", true);
    });
    },
    dataReceived: function
() {

        oViewModel.setProperty("/busy", false);
    }
    }
    });
    },
    _onBindingChange : function () {
        var oView = this.getView(),
            oViewModel =
this.getModel("objectView"),
            oElementBinding =
oView.getElementBinding();
        // No data for the binding
        if
(!oElementBinding.getBoundContext()) {

            this.getRouter().getTargets().display("objectNot
Found");

            return;
        }
        var oResourceBundle =
this.getResourceBundle(),
            oObject =
oView.getBindingContext().getObject(),
            sObjectId = oObject.Zoid,
            sObjectName =
oObject.Zpname;

        // Everything went fine.

```

```

oViewModel.setProperty("/busy",
false);

oViewModel.setProperty("/saveAsTileTitle",
oResourceBundle.getText("saveAsTileTitle",
[sObjectName]));

oViewModel.setProperty("/shareOnJamTitle",
sObjectName);

oViewModel.setProperty("/shareSendEmailSubject",

oResourceBundle.getText("shareSendEmailObjectSub
ject", [sObjectId]));

oViewModel.setProperty("/shareSendEmailMessage",

oResourceBundle.getText("shareSendEmailObjectMes
sage", [sObjectName, sObjectId, location.href]));
    }
    });
}
);

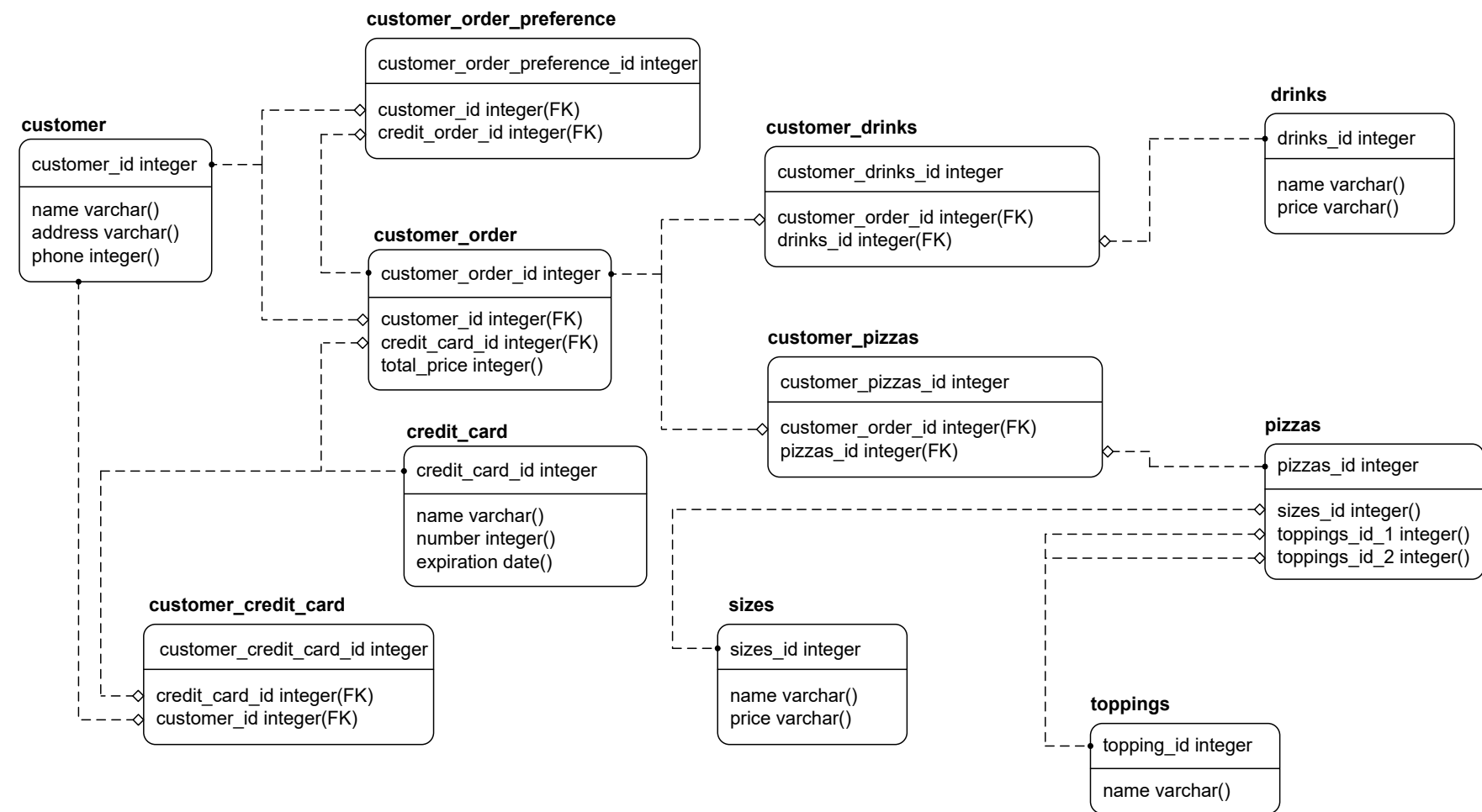
```

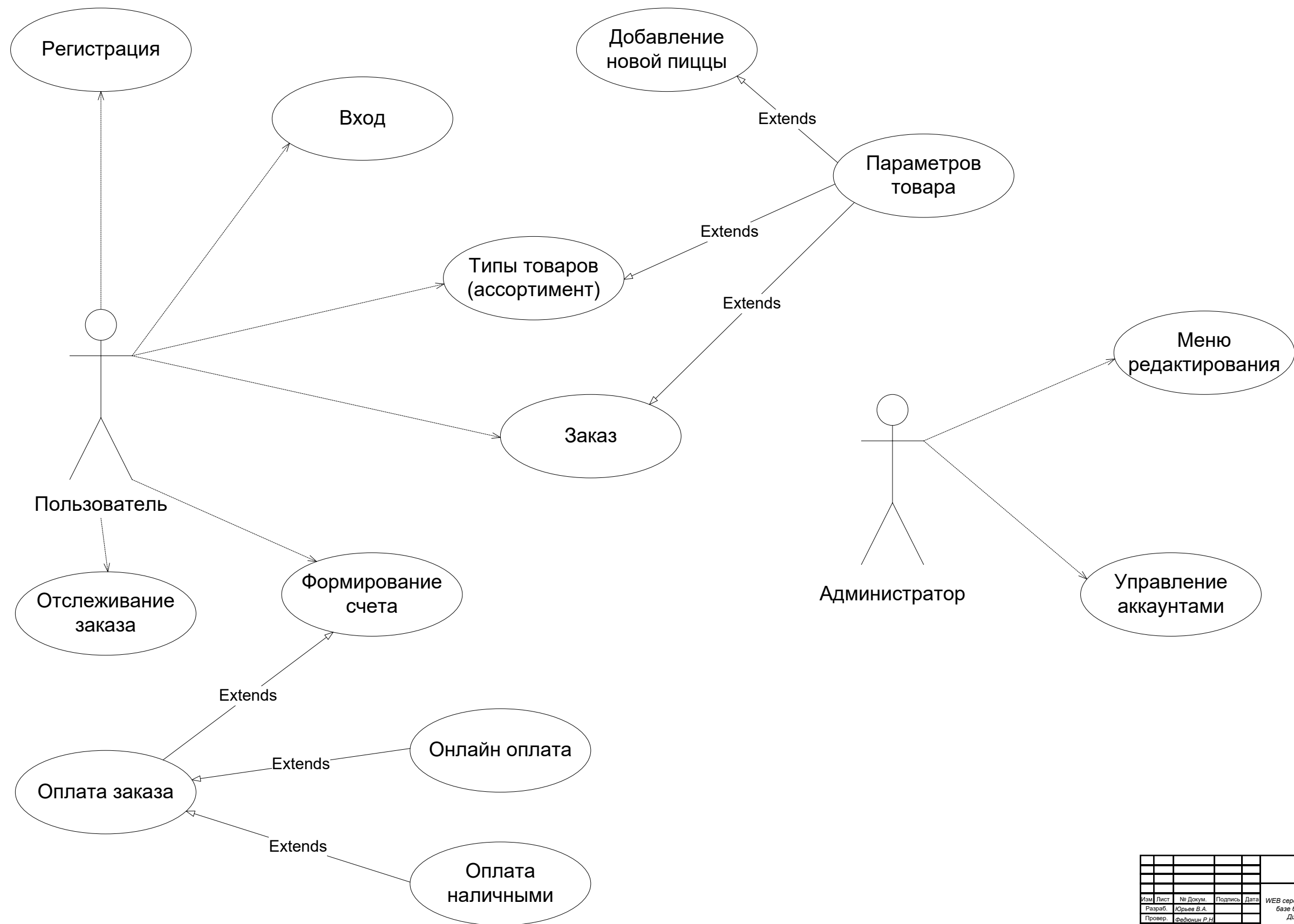
					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		117

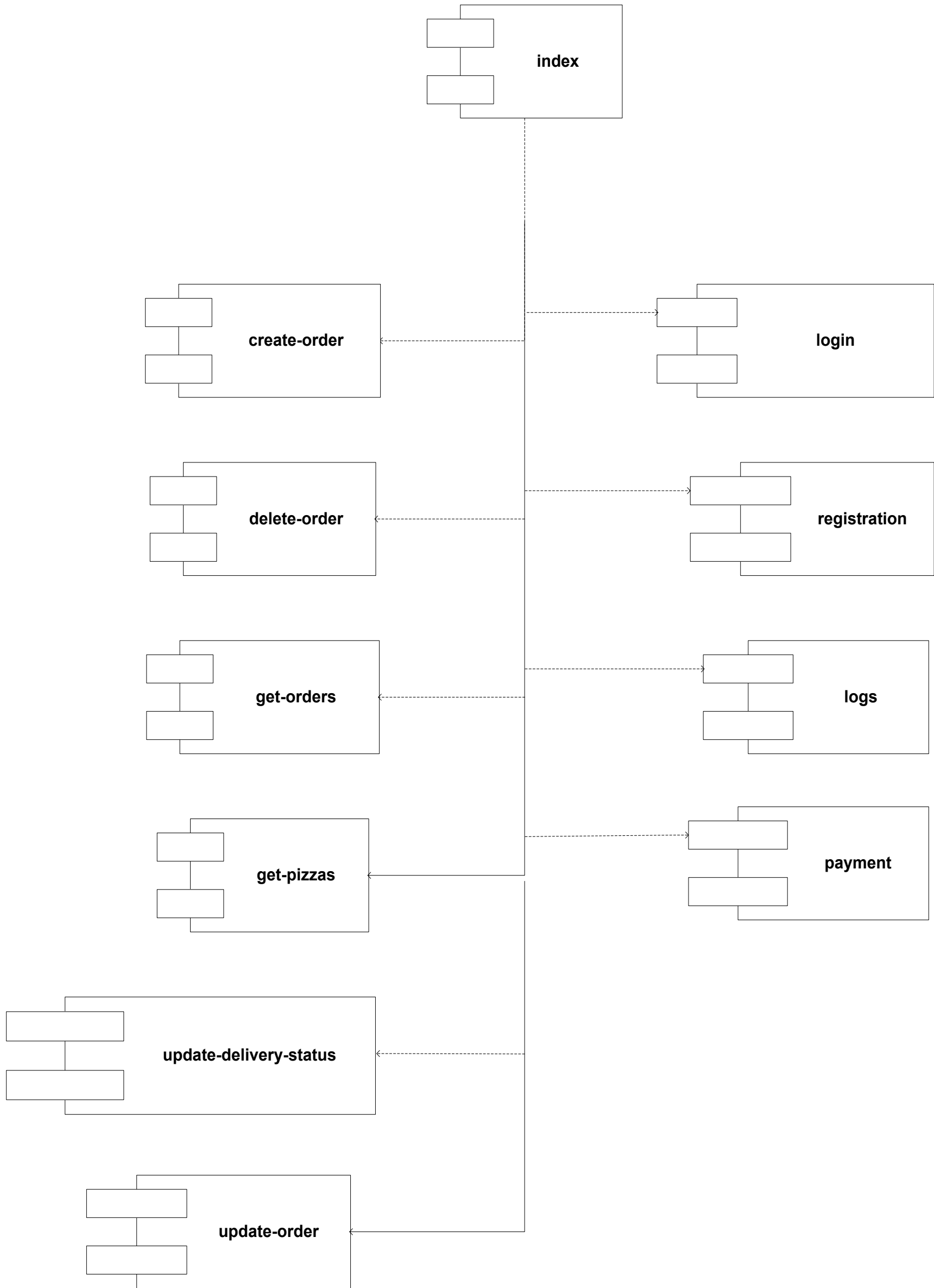
ПРИЛОЖЕНИЕ В

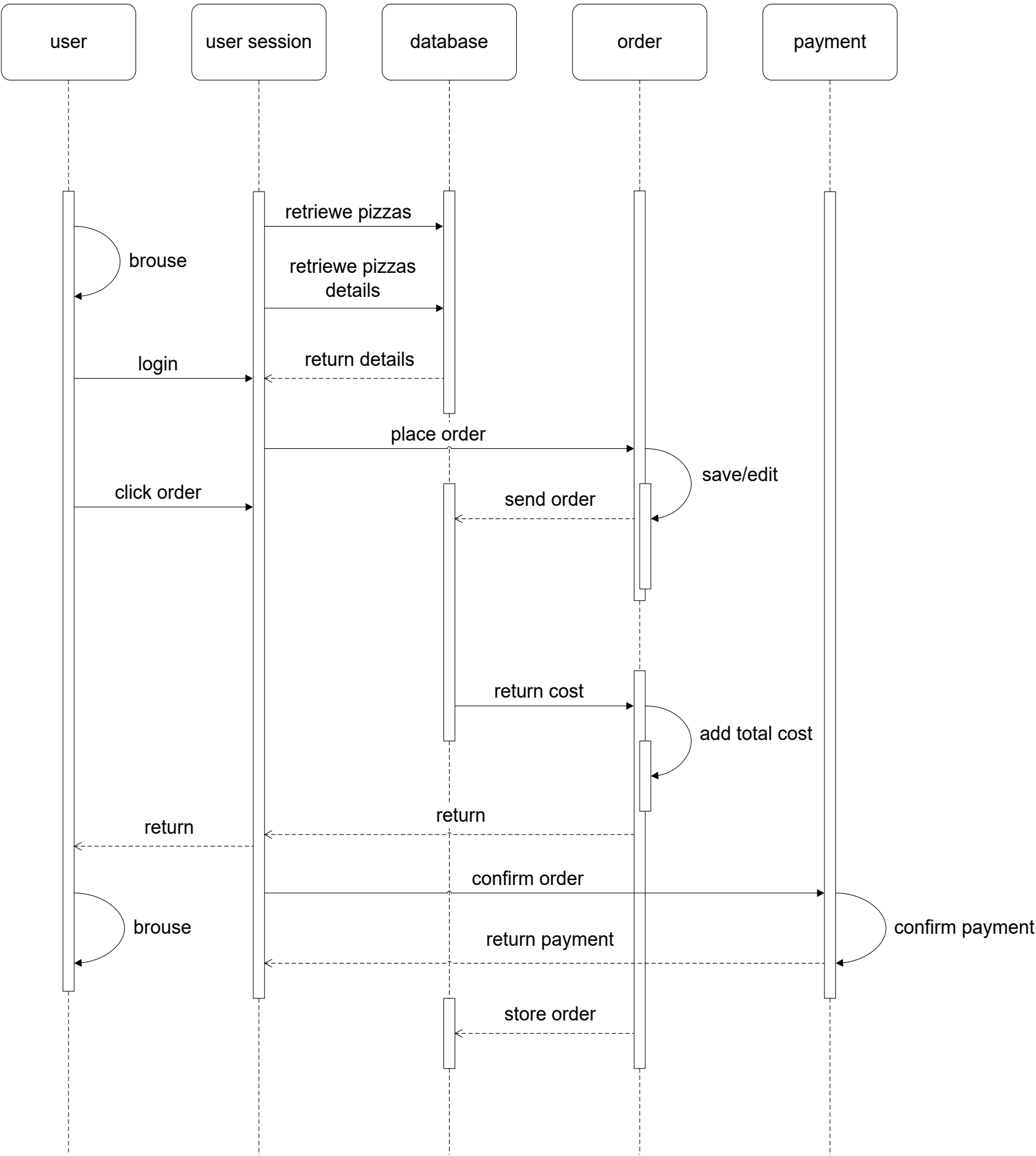
СХЕМЫ

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		118









ПРИЛОЖЕНИЕ С

ПРЕЗЕНТАЦИЯ

					ПГУ 1.090501.17.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		124

Изм.					
Лист					
№ докум.					
Подпись					
Дата					
ПГУ 1.090501.17.001 ПЗ					
Лист	125				

ТЕМА РАБОТЫ: **WEB-сервис электронной** **коммерции на базе** **бессерверной** **архитектуры**

Автор работы: Студент группы 15BO2 Юрьев В. А.
Научный руководитель: к.т.н., доцент Федюнин Р. Н.

Изм.						
Лист						
№ докум.						
Подпись						
Дата						
ПГУ 1.090501.17.001 ПЗ						
Лист	126					

ЦЕЛИ ЗАДАЧИ И ОБЪЕКТЫ РАБОТЫ

- ▶ Цель работы - разработать масштабируемую, отказоустойчивую систему продажи товаров на базе облачных технологий.
- ▶ Объектом разработки является программное приложение, обеспечивающее клиентам доступ к каталогу товаров и услуг, администраторам доступ к настройкам системы, а программистам к инструментам добавления нового функционала на базе облачного сервиса.
- ▶ В результате проведенной работы должен быть разработан проект, удовлетворяющий поставленной задаче, а именно, создан программный комплекс - «тонкий» клиент, бессерверная бизнес логика и хранилище, и разработаны соответствующие алгоритмы, реализующие возможность хранить, обрабатывать и доставлять до потребителя информацию в виде сообщений.
- ▶ Данное приложение рекомендуется использовать для массового продвижения товаров и услуг. Система может быть внедрена в сегменте электронной коммерции компании сегмента масс-маркет.

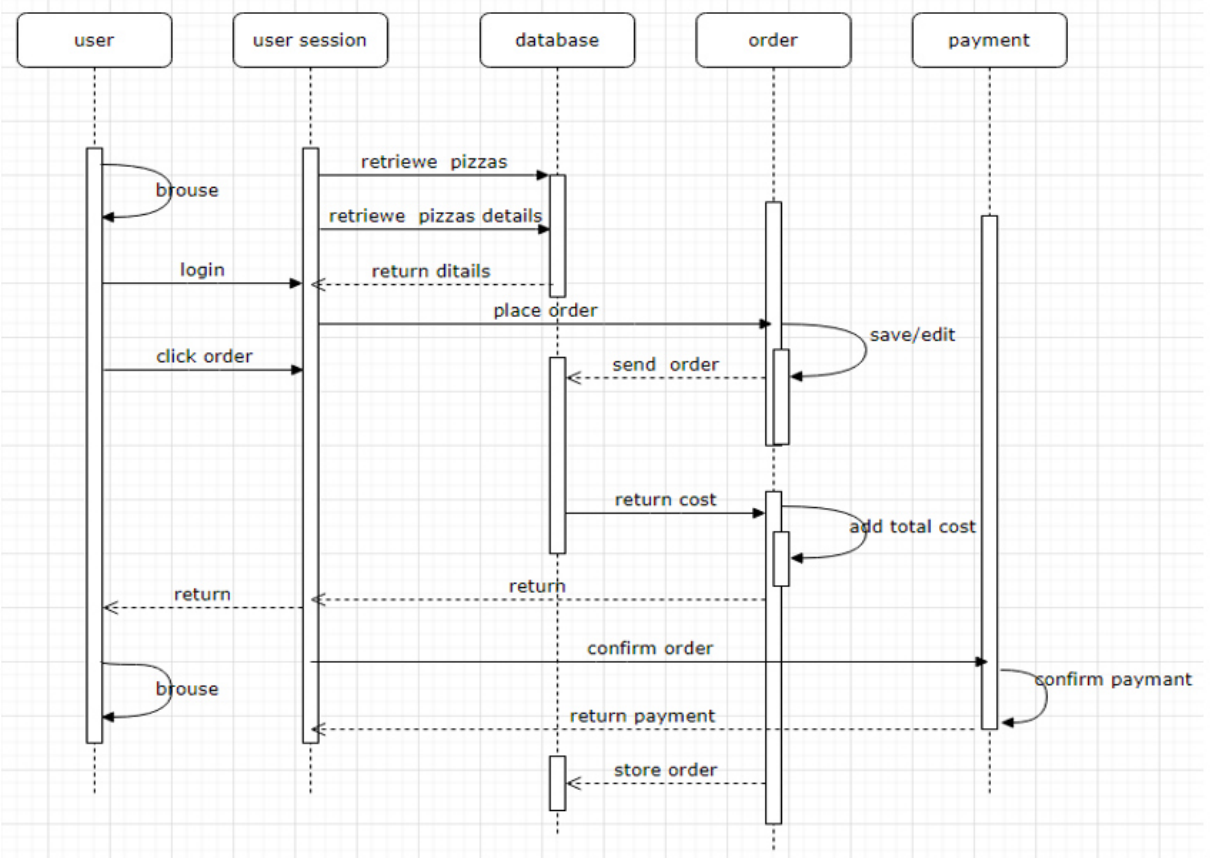
Изм.						
Лист						
№ докум.						
Подпись						
Дата						
ПГУ 1.090501.17.001 ПЗ						
Лист	127					

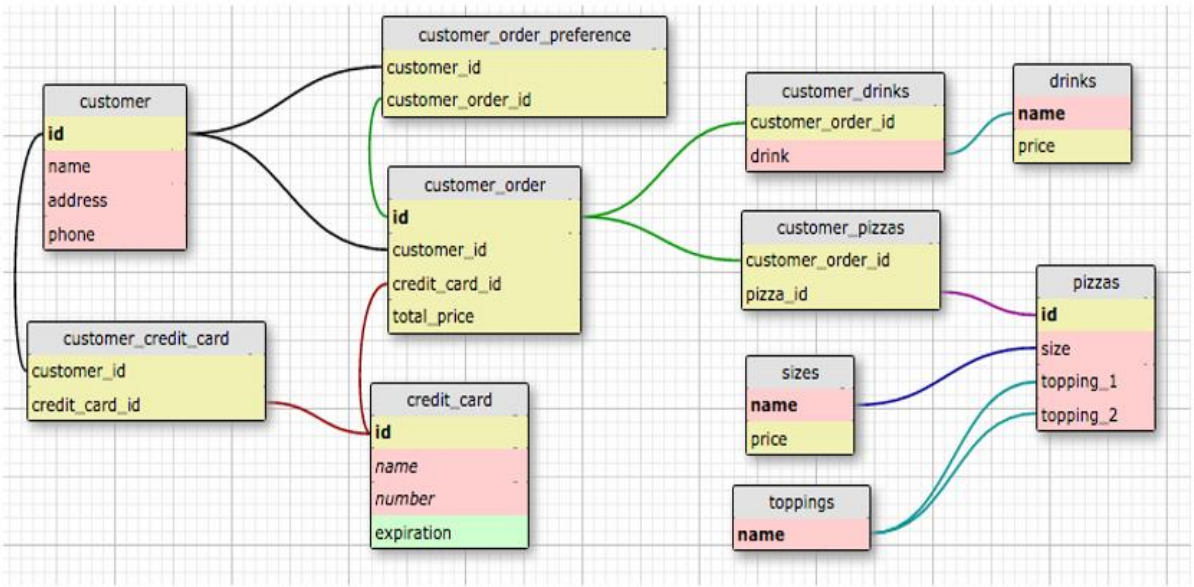
Что собой представляет WEB-сервис на бессерверной архитектуре

- ▶ Сегодня компании стали активно обращать внимание на «облака», думать об оптимизации и уплотнении вычислений на своих и облачных ресурсах, рассматривать пути сокращения расходов на ресурсы для вычислений.
- ▶ С развитием облачных сервисов вычисления становятся все более похожи на услугу, а потребителей (разработчиков) перестает интересовать физическая серверная инфраструктура, которая выполняет вычисления.
- ▶ Говоря о бессерверных вычислениях (serverless computing), нужно понимать, что на самом деле без физических и виртуальных серверов не обойтись ни при каких условиях, значит, речь идет не об отказе от аппаратной части, а о новом подходе к предоставлению сервисов в виде программных функций.
- ▶ Если задача команды разработчиков - сосредоточиться на бизнес-логике и выпускать продукты на рынок со скоростью света, то реализация проекта в рамках облачных сервисов отличный инструмент.
- ▶ Так и появляются на рынке бессерверные вычисления. Serverless в данном случае означает отсутствие задач/проблем по управлению инфраструктурой.

Традиционное IT	IaaS	Paas	SaaS	FaaS	
Приложения	Приложения	Приложения	Приложения	Приложения	<div> <div></div> - управляются разработчиками <div></div> - предоставляется как сервис </div>
Данные	Данные	Данные	Данные	Данные	
Среда исполнения	Среда исполнения	Среда исполнения	Среда исполнения	Среда исполнения	
Слой СПО	Слой СПО	Слой СПО	Слой СПО	Слой СПО	
ОС	ОС	ОС	ОС	ОС	
Виртуализация	Виртуализация	Виртуализация	Виртуализация	Виртуализация	
Серверы	Серверы	Серверы	Серверы	Серверы	
СХД	СХД	СХД	Серверы	Серверы	
Сеть	Сеть	Сеть	Серверы	Серверы	







Изм.	Пуст	№ докум.	Подпись	Дата
ПГУ 1.090501.17.001 ПЗ				
131	Пуст			

Вкусная еда в городе

Прием заказов ежедневно
с 10 до 23ч
4222-222

Пизза Десерты Напитки Дробовик Отзывы

0

Новинки

Пизца Пепперони.

Самая популярная пизца! Сaus Пепперони, Моцарелла, Томатный соус. Вес - 345г/330г.

маленькая (25см)

большая (33см)

235

в корзину

470

в корзину

Пизца Ассорти Гурман 33 см.

Пизца ассорти из 4 видов мяса: 1кг 2 сорта (Домашняя, Фарш, Шницель, Говядина). Вес - 880г.

большая (33см)

маленькая (25см)

в корзину

470

в корзину

Пизца Ассорти Классик 33 см.

4 вида мяса и сыра: 1кг 2 сорта (Пепперони, Пармезан, Карбонара, Говядина (по 2 сорта)). Вес - 880г.

большая (33см)

маленькая (25см)

в корзину

470

в корзину

Пизца Грибная

Вкусный грибной пизца с колбасой, шампиньонами, сыром, Моцарелла, Томатный соус, Черный соус. Вес - 320г/330г.

большая (33см)

маленькая (25см)

в корзину

470

в корзину

Вкусная еда в городе

Прием заказов ежедневно
с 10 до 23ч
4222-222

Пизца Десерты Напитки Дробовик Отзывы

235

Пизца

Пизца Пепперони.

Самая популярная пизца! Сaus Пепперони, Моцарелла, Томатный соус. Вес - 345г/330г.

маленькая (25см)

большая (33см)

235

1

в корзину

470

в корзину

Пизца Ассорти Гурман 33 см.

Пизца ассорти из 4 видов мяса: 1кг 2 сорта (Домашняя, Фарш, Шницель, Говядина). Вес - 880г.

большая (33см)

маленькая (25см)

в корзину

470

в корзину

Пизца Ассорти Классик 33 см.

4 вида мяса и сыра: 1кг 2 сорта (Пепперони, Пармезан, Карбонара, Говядина (по 2 сорта)). Вес - 880г.

большая (33см)

маленькая (25см)

в корзину

470

в корзину

Пизца Грибная.

Вкусный грибной пизца с колбасой, шампиньонами, сыром, Моцарелла, Томатный соус, Черный соус. Вес - 320г/330г.

большая (33см)

маленькая (25см)

в корзину

470

в корзину

Вкусная еда в городе

Прием заказов ежедневно
с 10 до 23ч
4222-222

Пизца Десерты Напитки Дробовик Отзывы

0

Пизца

Пизца Пепперони.

Самая популярная пизца! Сaus Пепперони, Моцарелла, Томатный соус. Вес - 345г/330г.

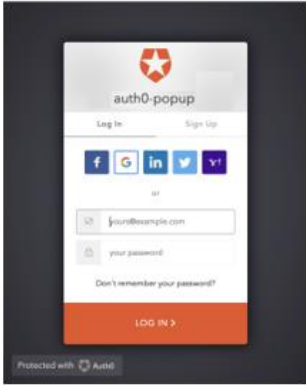
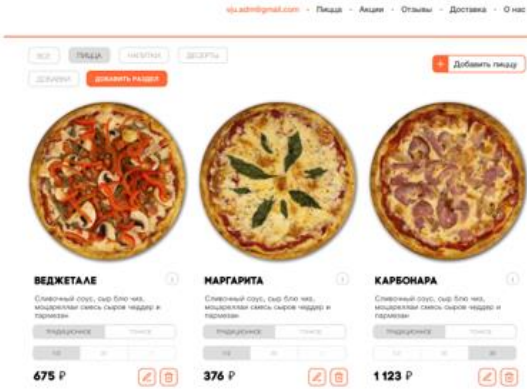
маленькая (25см)

большая (33см)

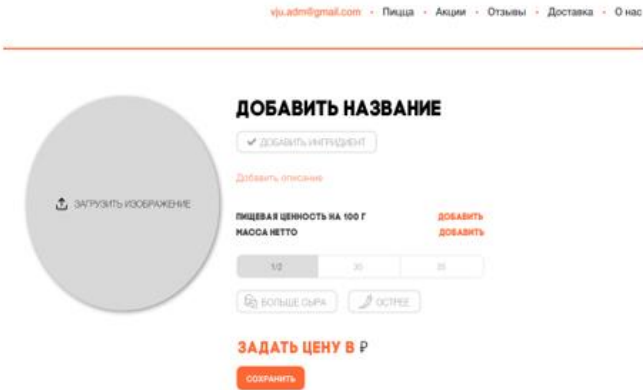
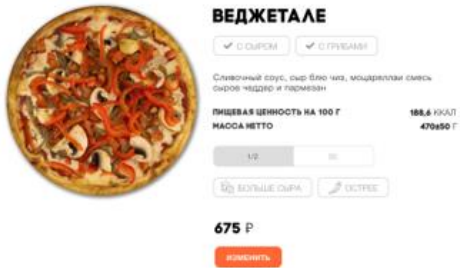
470

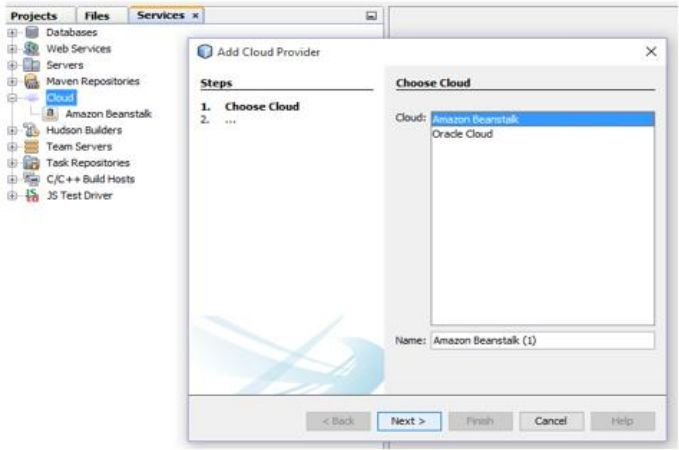
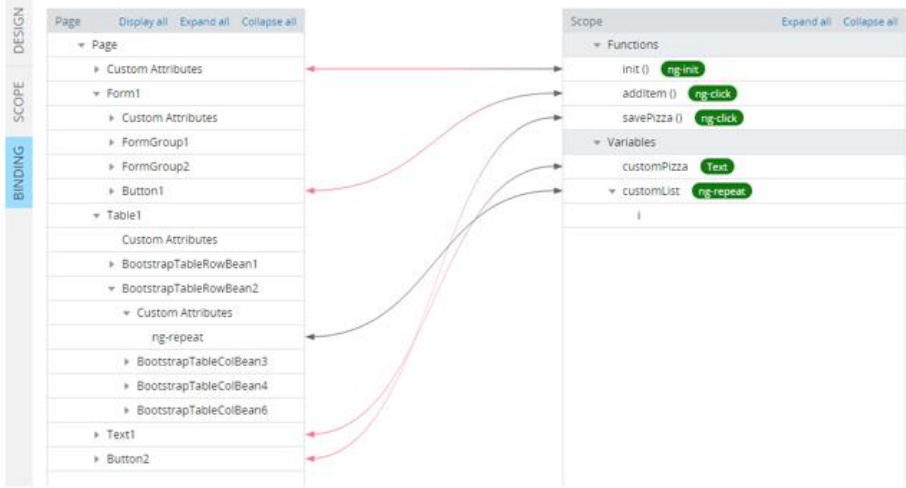
в корзину

Изм.	Лист	№ докум.	Подпись	Дата	ПГУ 1.090501.17.001 ПЗ	
					Лист	133



[vju.adm@gmail.com](#) · [Питца](#) · [Акции](#) · [Отзывы](#) · [Доставка](#) · [О нас](#)





Заключение

- ▶ В результате выполнения дипломного проекта был осуществлён анализ предметной области на команде разработчиков, изучены особенности её деятельности, определены основные требования к разрабатываемой автоматизированной системе, назначение и функции, было проведено её инфологическое и логическое проектирование, продумана архитектура системы, осуществлена её разработка, отладка и тестирование, подготовлена документация к системе.

Изм.	
Лист	
№ докум.	
Подпись	
Дата	

ПГУ 1.090501.17.001 ПЗ

Спасибо за внимание



					ПГУ 1.090501.17.001 ПЗ	Лист
						136
Изм.	Лист	№ докум.	Подпись	Дата		